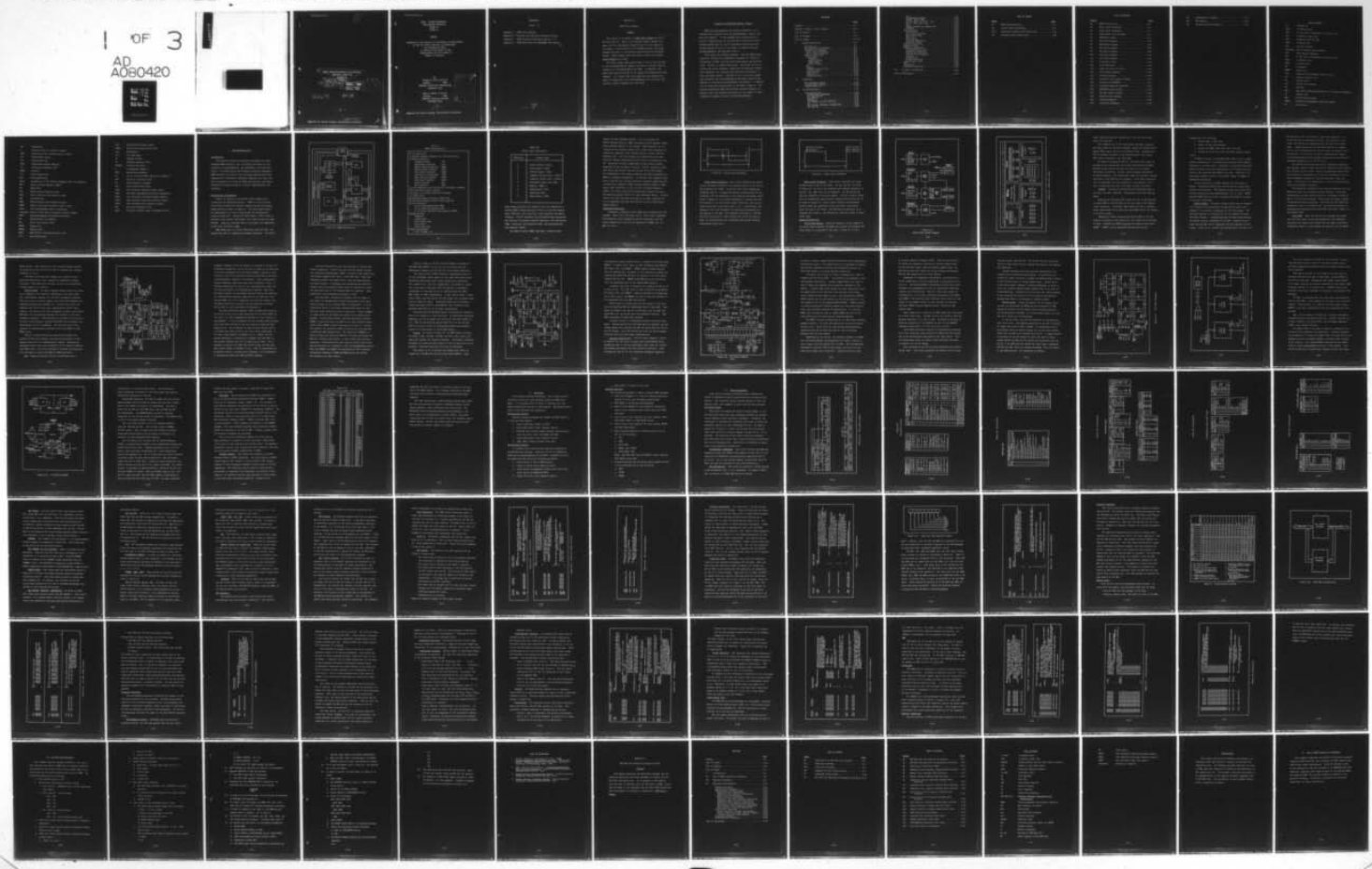
AD-A080 420 AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OH SCHOO--ETC F/G 9/2
MIME: MICROPROGRAMMABLE MINICOMPUTER EMULATOR. PHASE II. VOLUME--ETC(U)
DEC 79 T R HOYT, D A MYERS
UNCLASSIFIED AFIT/GCS/EE/79-11-VOL-2 NL

1 OF 3
AD
A080420



AFIT/GCS/EE/79-11

(1)

(6) MIME: MICROPROGRAMMABLE MINICOMPUTER EMULATOR. PHASE II - VOLUME II.

(9) Master's THESIS,

(14) AFIT/GCS/EE/79-11-VOL-2

Thomas R. Hoyt
Captain USAF

(10) Dean A. Myers
Captain USAF

(11) Dec 79

(12) 250

- X -

012 225

AFIT/GCS/EE/79-11

MIME: MICROPROGRAMMABLE MINICOMPUTER EMULATOR PHASE II VOLUME II

THESIS

Presented to the Faculty of the School of Engineering
of the Air Force Institute of Technology
Air University (ATC)
in Partial Fulfillment of the
Requirements for the Degree of
Master of Science

by
Thomas R. Hoyt, B.S.M.E.
Captain USAF
Graduate Electrical Engineering
December 1979

Dean A. Myers, B.S.E.E.
Captain USAF
Graduate Computer Systems
December 1979

-Y-

Approved for public release; distribution unlimited.

Contents:

Volume II

Appendix A

MIME User's Manual

Preface

This manual is an update of MIME User's Manual by Purvis and Yoho (Ref 1). Much of the original manual remains the same, and this information remains intact in this update of the manual. During Phase II of the MIME project, many small changes were made to the machine architecture and microword format. These changes have been made to bring the original User's Manual up to date.

The major change made during Phase II was to add sections in the microprogramming chapter to clarify, and give specific examples of, microprogramming the MIME. In addition, the page numbering and several of the tables and figures have been changed. It is hoped that these additions and changes will ease the transition into microprogramming for a new user and provide a useful reference for all users.

Preface to MIME Users Manual, Phase I

MIME (Microprogrammable Minicomputer Emulator) is an educationally oriented, user microprogrammable, general purpose minicomputer. It was designed and constructed as a master's thesis project at the Air Force Institute of Technology specifically for use in laboratory instruction and student design projects in the areas of computer control, microprogramming, and emulation (Ref 1).

This manual has several purposes: Describe MIME characteristics, provide the information necessary for effective utilization of MIME, and facilitate maintenance and modifications. Section I, System Description, introduces the architecture and system hardware, summarizes the possible information transfers, and concludes with detailed description of each functional module. Sections II and III provide operating procedures and microprogramming techniques respectively. Section IV concludes with maintenance information. A complete list of terms and abbreviations is included on page 77. The authors believe that MIME will provide valuable hands-on experience that will greatly contribute to the student's understanding of computer control and microprogramming.

Contents

List of Tables

List of Figures

List of Terms

AB	Address Bus
ALAT	A address latch
ALB	A less than B (Comparison of bytes A & B)
ALU	Arithmetic Logic Unit
AMD	Advanced Micro Devices
AMUX	A address mux
AUX	Auxiliary module
backplane	Wiring between edge connector
BAR	Base Address Register
BLA	B less than A (Comparison of Bytes A & B)
BLAT	B address latch
BMUX	B address mux
BP	breakpoint
BRMUX	Branch mux
CCR	Condition Code Register (macro level)
CCU	Computer Control Unit
CS	Control Store
CSB	Control Store Buffer
DB	Data Bus
DBR	Data Buffer Register:Buffers DB to D inputs of AM2901's
DC	Display code
DMA	Direct Memory Access
EPROM	Eraseable programmable read-only memory
FP	Front Panel

hex	Hexadecimal
i	Subscript used to indicate integer
IMUX	Instruction mux: chooses input to 2901's
I/O	Input/Output module
IOB	Input/Output Bus
IOBR	Input/Output Buffer Register
IR	Instruction Register (CCU)
KYBRD	Keyboard
L.S.	Least Significant
MAB	Microaddress Bus
macro	Designation of the machine language level of operation
MAR	Memory Address Register (MEM1)
MB	Memory Bus
MBR	Memory Buffer Register (ALU)
MCCR	Micro Condition Code Register (ALU)
MDB	Microdata Bus
MEM1	Memory module: main program storage
MEMFF	Memory flip-flop: specifies read or write
micro	The microprograms level of operation
microword	Group of bits used to generate control signals
MIME	Microprogrammable Minicomputer Emulator
MR	Mask Register (in Am2914)
MLC	Micro Loop Counter (CCU)
MPMUX	Mapping mux
MPROM	Mapping PROM
MSKB	Mask Buffer: buffers PL 63-56 to DB
M.S.	Most Significant

MSP	Miscellaneous Signal Panel
MTCM	Micro Test Condition Mux (ALU)
mux	Multiplexer
N.A.	Not Applicable
PC	Program Counter
PL	Pipeline Register (CCU)
POLMUX	Polarity Mux (CCU)
Q	Q registerin Am2901's
RAM	Random Access Memory
Ri	One of 16 2-port RAM locations in Am2901's
DFF	Direction flip-flop (I/O)
SR	Status Register (Am2914)
TCM	Test Condition Mux (ALU)
TCRB	Test Condition Register Buffer (ALU)
UART	Universal Asynchronous Receiver-Transmitter
UDACi	User Definable Auxiliary Command number i
UDAFi	User Definable Auxiliary Function number i
UDTCi	User Definable Test Condition number i
WCR	Word Count Register (I/O)
YBFR	y Buffer: buffers output of Am2901's to DB

1. System Description

Introduction

This section presents information necessary for understanding MIME operation, and it provides the basis for the operating, microprogramming, and maintenance sections which follow. The Architecture and Hardware subsection addresses MIME structure in terms of the functional modules and their organization; Information Transfers summarizes the data flow that is possible utilizing the MIME architecture; and Module Descriptions presents details of module implementation and operation.

Architecture and Hardware

Figure A1 contains a functional block diagram which illustrates the MIME architecture. MIME is a bus oriented, functionally modularized digital computer having the general characteristics listed in Table AI. The functional modules are implemented on circuit cards having the designations listed in Table AII. Except for Front Panel 1 (FP1) which is affixed to the cabinet front panel, all circuit cards are rack mounted. Terms and acronyms used in the manual are summarized in the list of terms on page

MIME makes use of a 16 bit data word, Data Bus (DB), and Address Bus (AB) for register and memory transfers. The front

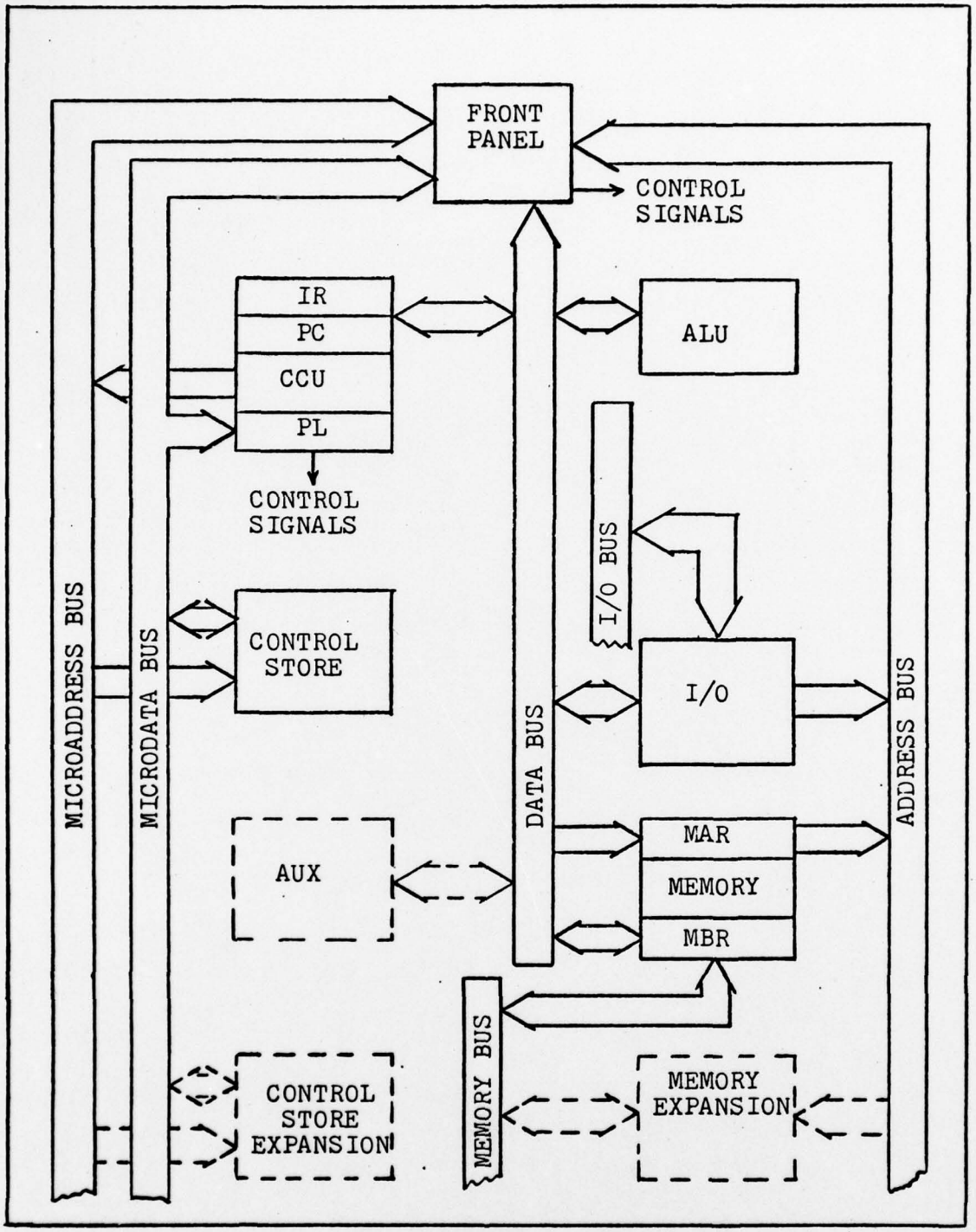


Figure A1. MIME Architecture

Table AI

MIME Characteristics

16 bit data word

17 general purpose registers (Q, R0 through R15)

Dedicated registers

- Program Counter (PC)
- Instruction Register (IR)
- Memory Address Register (MAR)
- Memory Buffer Register (MBR)
- Data Buffer Register (DBR)
- Condition Code Register (CCR)
- Base Address Register (BAR)
- Word Count Register (WCR)
- Status Register (SR)
- Mask Register (MR)

16 user definable functions/16 user definable commands

64K words main memory capacity

16 levels of vectored priority interrupt

1.43 MHz clock

Synchronous memory reference operations

4K microword maximum Control Store capacity

64 bit microword

- 12 bit microaddress bus (MAB)
- 64 bit pipeline register (PL)
- 8 bit micro condition code register (MCCR)

Operating modes

- automatic
- single step
- auto single step
- micro step
- auto micro step

Power requirements

- 110V, 50-60Hz, 2A

Table AII

Circuit Card Identifiers

Identifier	Circuit Card
A	Control Store 2 (CS2)
B	Front Panel 2 (FP2)
C	Control Store 1 (CS1)
D	Computer Control Unit 1 (CCU1)
E	Computer Control Unit 2 (CCU2)
F	Arithmetic Logic Unit (ALU)
G	Memory 1 (MEM 1)
H	Input/Output (I/O)
I	Auxiliary (AUX)
P	Front Panel 1 (FP1)

panel module provides the operator with the capability to control MIME's operating mode as well as load and display macro (Machine) level and micro level register and memory contents. The ALU performs the arithmetic/logic operations and contains the general purpose registers, the Data Buffer (DBR), Condition Code Register (CCR), and Microcondition Code Register (MCCR).

The memory module (MEM1) provides a random access

memory for main program storage. It also contains the Memory Address Register (MAR) and Memory Buffer Register (MBR) which provide buffers to the busses. MIME operation is controlled by the Computer Control Unit (CCU) which contains the Instruction Register (IR), Program Counter (PC) and Pipeline Register (PL). The CCU decodes the instruction which came from the location specified by the PC and is contained in the IR. Execution is then accomplished by the control signals generated by the contents of the PL. The PL contains a microword which was fetched from the Control Store (CS) module. The Input/Output (I/O) module provides the means necessary for MIME to interface peripherals. It contains the hardware necessary to provide the vectored priority interrupt capability and the Word Count Register (WCR) and Base Address Register (BAR) used in DMA transfers, as well as the I/O Buffer Register (IOBR), I/O device selection decoding and a serial I/O port. The Auxiliary Module (AUX) is a user definable module. Various control and data lines are provided at an edge connector for use in expanding the MIME architecture to suit a particular target machine.

Information Transfers

Information transfers within MIME may be divided into two classes: Micro level and macro level. Micro level transfers are defined as those between CS and CCU and macro level as those between any of the 16 bit registers of MIME (e.g. IR, MBR, R0, etc.)

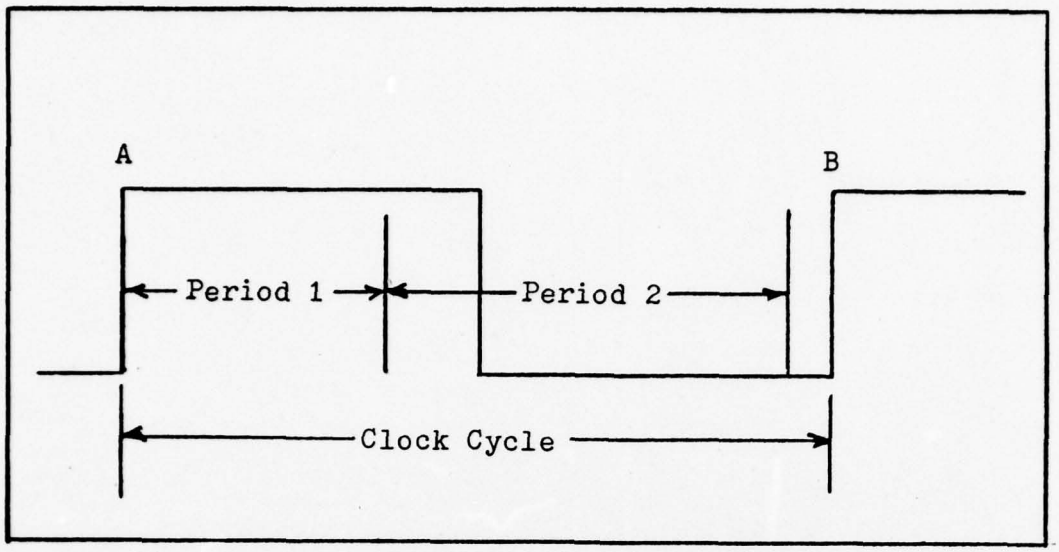


Figure A2. Micro Level Transfers

Micro Level Transfers. Micro level transfers are essentially successive fetches of microwords from CS to the PL via the microdata Bus (MDB). Using the first level pipelining technique (Ref 2), a single microword is fetched from CS concurrently with the execution of the previous microword. Referring to Figure A2, the current microword is clocked into the PL at point A, the clock low-to-high transition, and is executed during the clock cycle from A to B. Concurrently, the address of the next microword is calculated during period 1 and applied to the MDB. The addressed microword is fetched from CS during period 2 and clocked into the PL at point B. Thus, one microinstruction is executed and another fetched during each clock cycle.

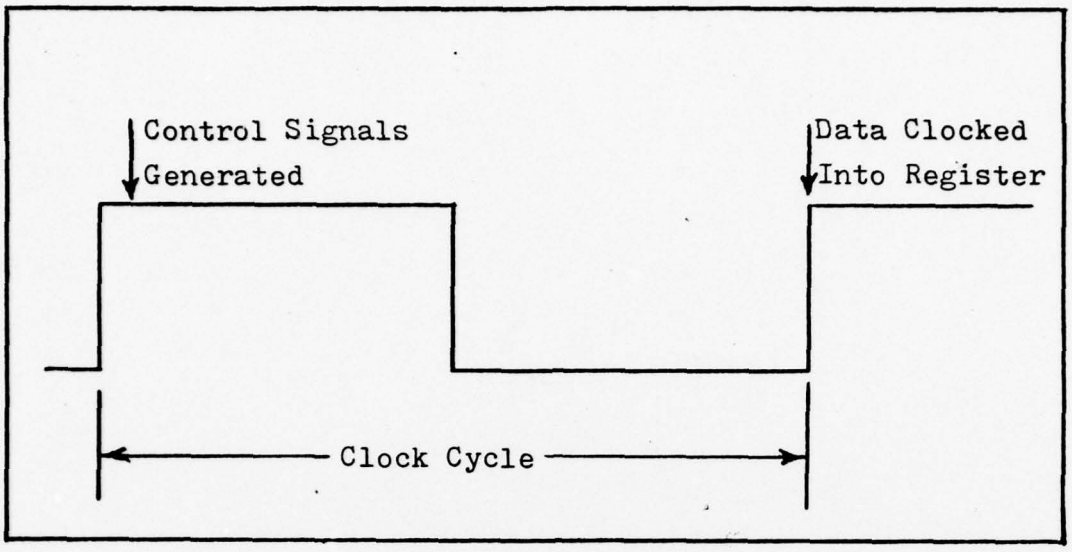


Figure A3. Macro Level Transfers

Macro Level Transfers. Macro level transfers are accomplished using three 16 bit buses: DB, AB, and MB. As shown in Figure A1, the DB is the transfer path between registers in the CCU, ALU, MEM, AUX, and I/O modules. The MB and AB are used for transfers to and from memory. Control of macro transfers is accomplished using control signals derived from the PL. As shown in Figure A3, these control signals are available shortly after the beginning of any particular clock cycle until the end of the cycle. This allows ALU functions to be computed and routed to the destination register during a single clock cycle.

Module Descriptions.

Front Panel Module. Figure A4 depicts a block diagram of the Front Panel designed for MIME, while Figure A5 presents the Front Panel as it appears to the user. Volume III of this

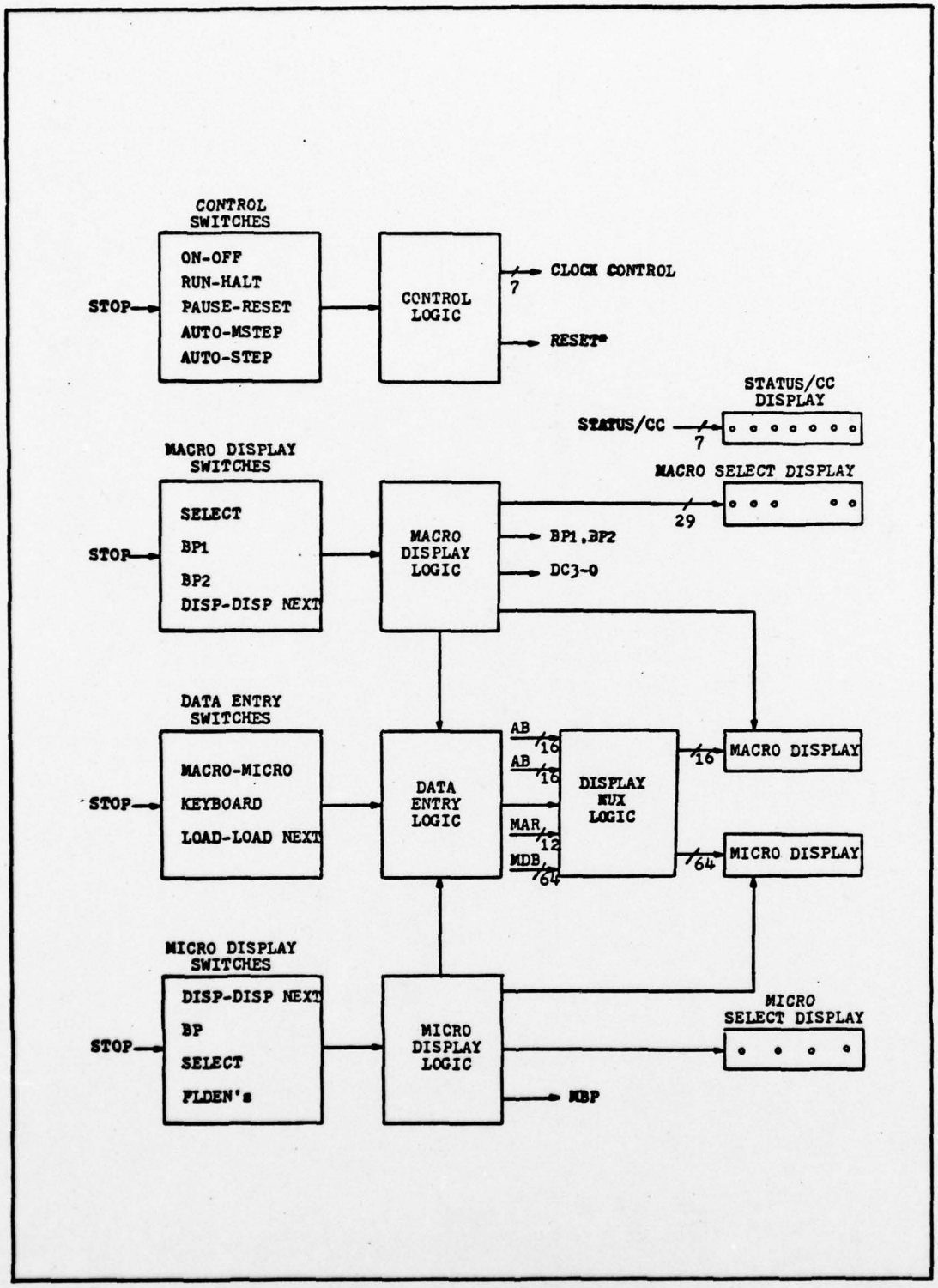


Figure A4.
Front Panel Block Diagram

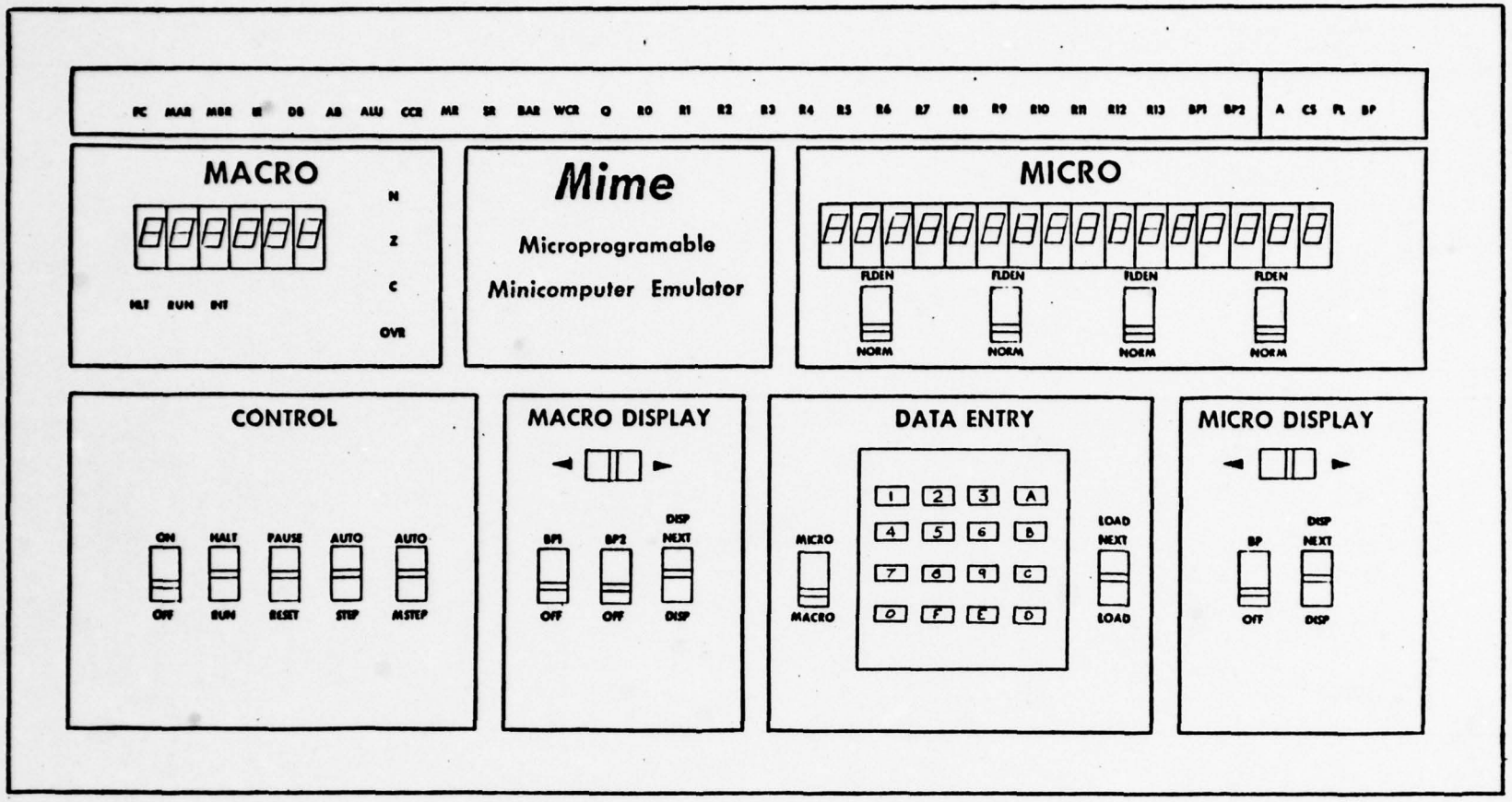


Figure A5. MIME Front Panel

report contains detailed schematics of the two front panel cards, FP1 and FP2.

FP1 contains all of the front panel seitches, displays, and logic shown in the block diagram, except the Display Multiplexer (MUX) logic, which is contained in FP2. Consequently, FP1 is attached directly to the hinged chassis front panel, while FP2 is mounted in the card rack.

As shown in Figures A4 and A5, the Front Panel logic is grouped in the same manner as the FP1 switches and displays. Separate logic modules are used for each of the primary front panel functions: Control, Macro Display, Data Entry, and Micro Display. The Display Mux logic is provided to allow selection of any MIME bus for display. Front panel operation will be discussed from the viewpoint of the specific occurrences resulting from each front panel switch activation.

CONTROL. Turning the ON-OFF switch ON applies power to the built in power supplies. Turning the switch OFF removes power.

Momentarily depressing RUN causes the CCU to synchronously place MIME in the RUN mode on the next high-to-low master clock transition. Similarly, HALT causes the CCU to synchronously place MIME in the HALT mode when MIME enters the instruction fetch routine.

Momentarily PAUSE synchronously halts MIME on the next high-to-low master clock transition, and resets the clock mode to fast. Momentarily RESET generates the master clear pulse RESET*. RESET* is also generated during power-up and

accomplishes the following:

1. Places MIME in HALT mode
2. Loads initials microaddress
3. Clears PC, MEMFF, MAR, WCR, BAR, CCR, DBR
4. Selects PC and A for macro and micro display resepctively

If MIME is halted, a momentary STEP causes only a single machine instruction to be fetched and executed before MIME is returned to the HALT state. Similarly, a momentary AUTO STEP causes successive machine instructions to be fetched and executed at the selected AUTO MSTEP clock rate. MIME will then halt under program control or when HALT, PAUSE, or RESET is momentarily activated.

Initial actuation of MSTEP changes the clock mode to manual. Subsequent MSTEP's can then be used to single-step through microroutines. ATUO MSTEP provides clock rates which are user selectable from approximately 1 Hz to 1 kHz. MSTEP and AUTO MSTEP should not be activated while MIME is running in the fast clock mode.

MACRO DISPLAY. The Macro Display Select switch designated in Figure A5 by <> selects the particular macro level register to be loaded or displayed by the front panel. The selected register is indicated by the illuminated LED above the macro display. Loading/displaying the MBR loads/displays the memory location addressed by the MAR. Loading the ALU loads the ALU DBR, while displaying the ALU displays the ALU output. Since Q, Ri, and BPi are located within the ALU, the

ALU DBR must first be loaded to load these registers. Q, Ri and BPi may be displayed directly. For operator convenience, R14 and R15 are designated BP1 and BP2 on the front panel. Identification of the selected register is encoded into a four bit Destination Code (DC) which is routed to the CCU and ALU. A fifth bit is used to differentiate between the sixteen Am2901 general purpose registers and the discrete dedicated registers (e.g. IR, PC, etc.).

BP1 and BP2 enable the macro level breakpoint registers. When enabled, MIME is halted by the CCU whenever the PC matches the contents of BP1 or BP2. This is accomplished under firmware control by comparing the PC with the contents of the enabled breakpoint register(s) and halting MIME if they are equal. This action only takes place if the user has provided microcoded breakpoint handlers.

DISP gates the contents of the selected register onto the DB/AB and then into the macro display register. DISP NEXT accomplishes the same function and also increments the MAR if MBR is the selected register. This allows convenient display of sequential memory locations. Both DISP and DISP NEXT are operational only in the HALT mode. The selected register is automatically displayed after each STEP, AUTO STEP, MSTEP, or AUTO MSTEP CYCLE.

DATA ENTRY. Macro and micro level programs are loaded from the front panel using the hexadecimal (hex) KEYBOARD. Successive hex digits are shifted into the least significant (rightmost) macro or micro display as specified by the MACRO/

MICRO switch. Each character of the selected display shifts one position to the left as new data is entered; the leftmost character is last.

Contents of the selected display are loaded into the specified macro/micro level register by momentarily depressing LOAD. LOAD NEXT does the same, as well as incrementing the MAR (macro) or A (micro).

MICRO DISPLAY. The Micro Display switch selects the micro level register to be loaded or displayed by the front panel. The microaddress register (A) and micro breakpoint register (BP) are 12 bits long and require only three hex displays; the remaining displays are blanked out. Control Store (CS) and Pipeline (PL) are 64 bits long, requiring the entire micro display. BP controls the micro breakpoint feature, which halts MIME when the microaddress matches the contents of the micro breakpoint register. DISP displays the contents of the selected micro level register, whereas DISP NEXT also increments the microaddress if CS is displayed. The selected micro level register is automatically displayed after each MSTEP or AUTO MSTEP cycle.

Each of the FLD ENA switches selectively enables one quarter of the micro display, allowing any quarter(s) of the display (and CS) to be independently changed. Enabling any particular field (four hex characters) of the display disables each NORM field. The entire display is enabled by placing all four switches in either the NORM or FLDEN positions.

ALU. Figure A6 depicts the ALU configuration and a

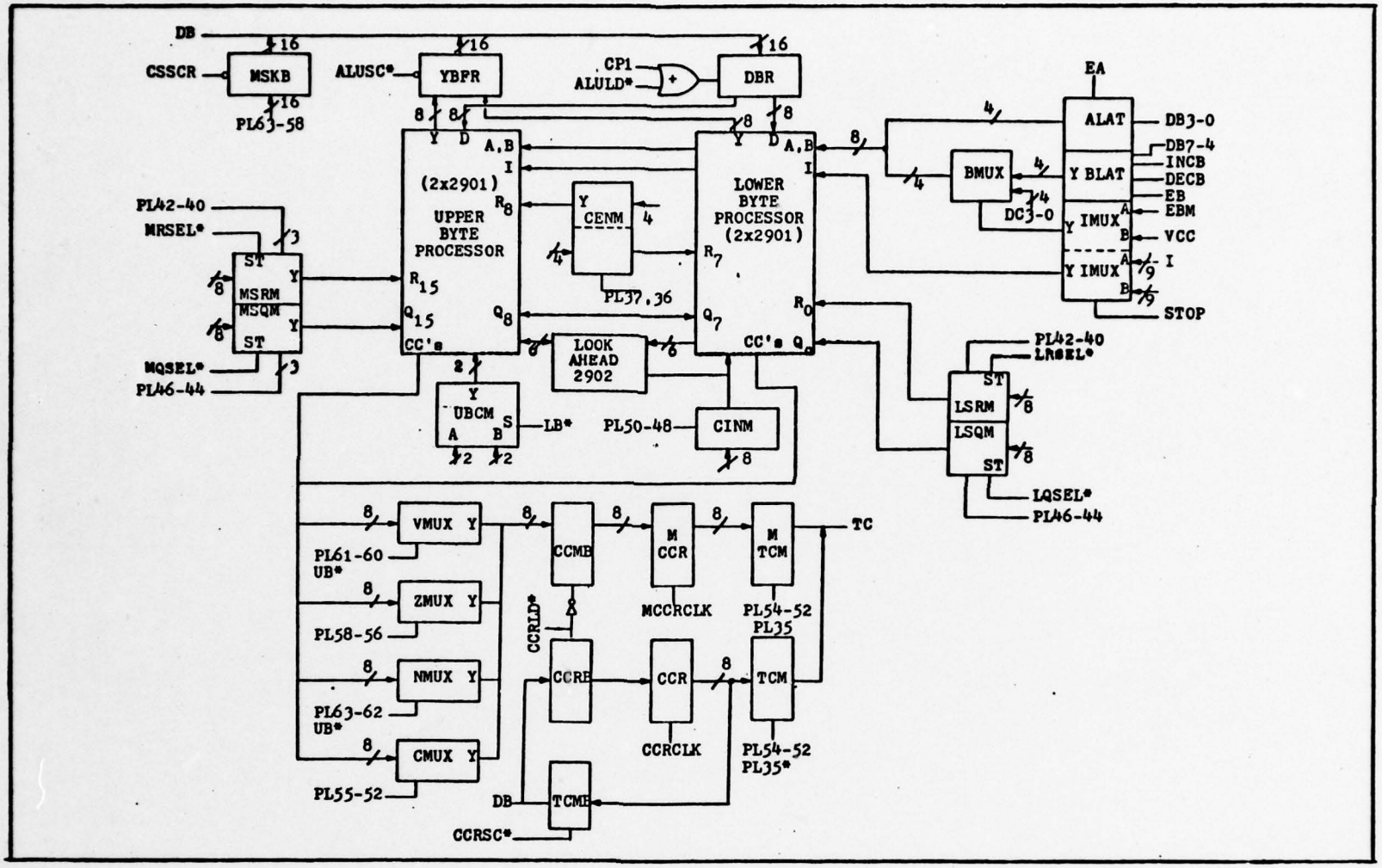


Figure A6. ALU Block Diagram

schematic drawing of the ALU module is included in Volume III. As shown in Figure A6, the 16 bit ALU is based on two identical 8 bit byte processors with shifting linkages, condition code selection, and other ancillary functions provided by associated multiplexers, registers, and buffers. This configuration allows autonomous upper and lower byte arithmetic, logic, or shifting operations as well as full word operations when the two byte processors are simultanesouly enabled. Each byte processor consists of two Am2901 four-bit bipolar microprocessor slices having combined resources of a 16 word by 8 bit two-port RAM, an 8 bit Q register, an 8 bit ALU element, and associated shifting, decoding, and multiplexing circuitry. The Am 2901 is fully described in Reference 3.

The ALU Data Buffer Register (DBR) accepts data from the 16 bit DB on the low-to-high transition of CP1 when ALULD* is low, and it is then immediately available at the D inputs of all Am2901's. A and B addresses for the two-port RAM are available from DB3-0 and DB7-4, and are latched by ALAT and BLAT respectively. The A address is selected from the DB, and it is latched into ALAT by EA on the low-to-high transitions of CP1 when either ALTLD* or ABLTLD* is low. BMUX1 (see CCU module description) selects the B address from the BLAT or front panel Display Code (DC) as specified by STOP. The B address is latched into BLAT from DB7-4 during the RUN mode when either BLTLD* or ABLTLD* is low. BLAT is implemented as an up/down counter, allowing the B address to be incremented or decremented using the INCB and DECB commands.

Nine microinstruction bits are required to control processor operations. These bits enter the ALU module through the Instruction Multiplexer (IMUX) to provide front panel load and display capabilities while in the STOP mode. Both byte processors receive the same control bits and hence perform the same function. Autonomous byte operations are possible because only the processor whose clock is enabled will store the resultant data in its internal RAM or Q register. Data generated by the nonenabled processor is lost.

Byte processor construction allows a one bit right or left shift of computational results during each clock cycle before storage in the internal RAM or Q registers. Ri or Qi connections serve as tri-state bidirectional ports for the most and least significant bits of each byte. Shift linkages to these ports are provided by an array of tri-state multiplexers. During right shifts, the Most Significant RAM Mux (MSRM) selects the source of RAM bit 15 while the Most Significant Q Mux (MSQM) selects the source of Q bit 15. During left shifts, the sources of RAM bit 0 and Q bit are selected by LSRM and LSQM respectively. The Center Mux (CENMUX) is used to select the source of RAM bit 8 during left shifts and RAM bit 7 during right shifts. Q bits 7 and 8 are directly connected. Microinstruction bits used to control shift linkage multiplexers are discussed in the microprogramming section. Note that MRSEL* and MQSEL* are generated by CCU to prevent inadvertant enabling of MSRM and MSQM during left shifts and likewise for right shifts.

Data is output to the DB from the Am2901 Y outputs via the YBFR when ALUSC* is low, and it is clocked into the destination register by the next CP1 low-to-high transition.

The Carry-In Mux (CINM) selects an appropriate value to be the look-ahead carry inputs for the upper byte during word operations and ripple carry inputs during byte operations. The overflow (V), zero (Z), negative(N), and carry (C) condition codes may be selected from the results of arithmetic operations, or they may be set or cleared using the VMUX, ZMUX, NMUX, or CMUX multiplexers. A less than B (ALB), B less than A (BLA), and the LSB of the ALU output (Y0) are also available as condition codes. Conditions codes may be stored in either the macro (CCR) or micro(MCCR) level condition code register when PL35 is low or high respectively.

Multiplexers TCM and MTCM select the particular condition code to be output as a test condition. To facilitate interrupt handling, the CCR is gated onto the DB by TCRB when CCRSC* is low. MSKB is provided for gating PL63-56 on to the DB when CSSC* is low as a means of transferring masks and constants stored in the microword to other MIME modules.

Memory. As shown in Figure A7, the MEM1 module contains 1K words of RAM, the MAR, the MBR, the byte write logic, and auxiliary command and function decoding. Information transfers necessary for read and write cycles in the run mode will be explained. Stop made operation will also be discussed.

Data which is to be written into memory must first be loaded into the MBR from the DB using signal MBRLD*. Next,

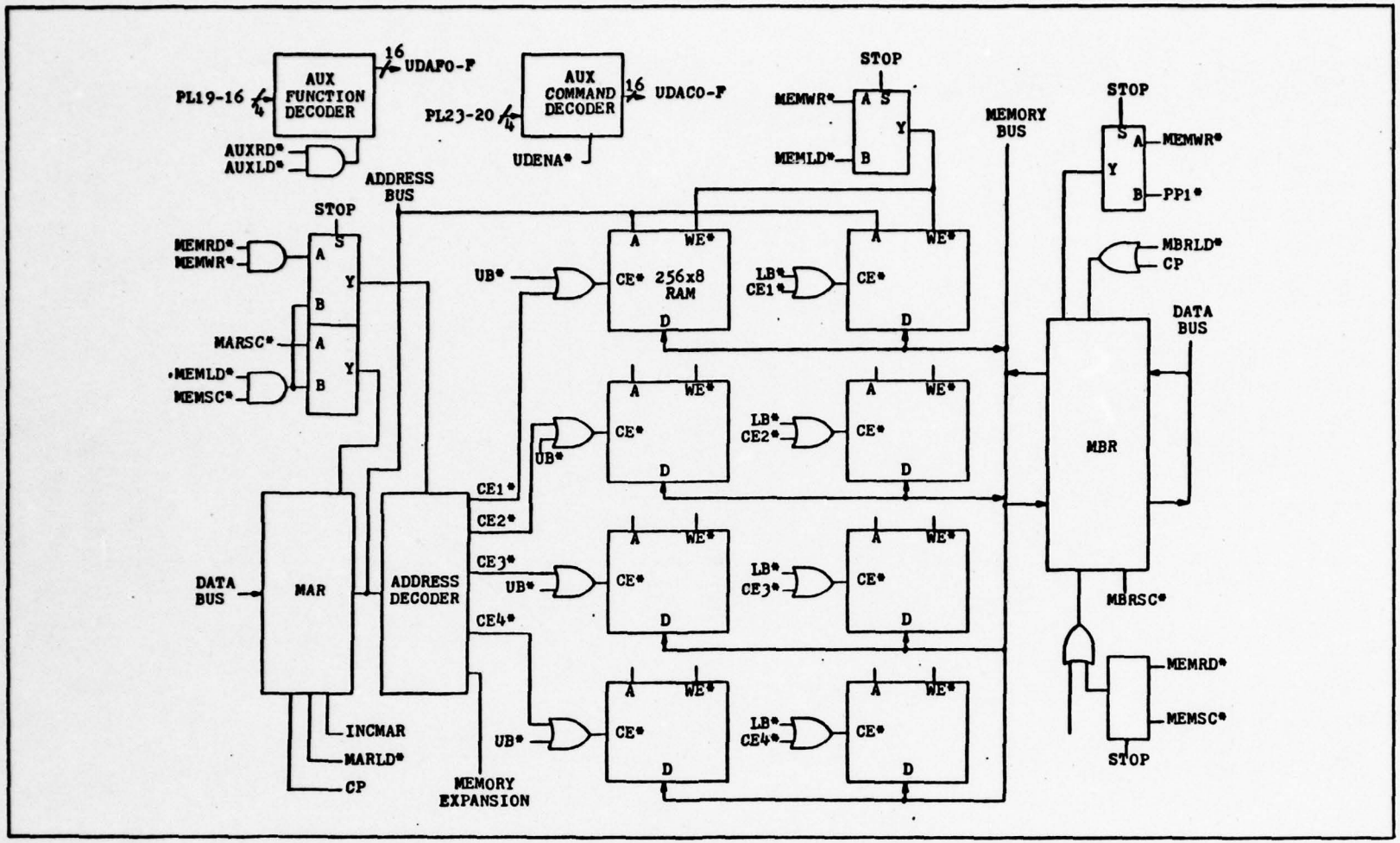


Figure A7. MEM1 Block Diagram

the effective memory address must be loaded into the MAR using MARLD*. A memory write cycle is then initiated using MARSC*, UB* and/or LB*, and MEMWR*. MARSC* gates the MAR contents onto the address bus, allowing it to be decoded to enable the appropriate 256x8 RAM block(s) as indexed by the byte enable signals UB* and LB*. MEMWR* gates MBR data onto the MB and into the addressed word of the enabled RAM block(s).

A memory read cycle is preceded by loading the MAR as in a write cycle. The read cycle is then initiated using MARSC*, UB* and/or LB*, and MEMRD*. Address decoding is accomplished as in write cycle, and MEMRD* is used to lead the contents of the addressed work into MBR from the MB.

Memory data is loaded from the front panel in the stop mode by selecting MBR and depressing LOAD or LOAD NEXT. This gates an address onto the AB, clocks data into the MBR, and gates MBR onto the MB and into the addressed location. LOAD NEXT increments the MAR when released.

A memory location is similarly displayed from the front panel. Depressing DISP or DISP NEXT gates an address onto AB, clocks memory data from MB into MBR and gates MBR onto the DB for FP display. Both UB* and LB* are held low while stopped allowing only full word memory manipulations.

Computer Control Unit. The CCU block diagram of Figure A8 depicts the CCU which is based upon Am2911/AM2909 microsequencers (Ref 2) and has associated logic used for initiating address generation and test condition selection. Also included are the IR, PC, PL, and Micro Breakpoint Register

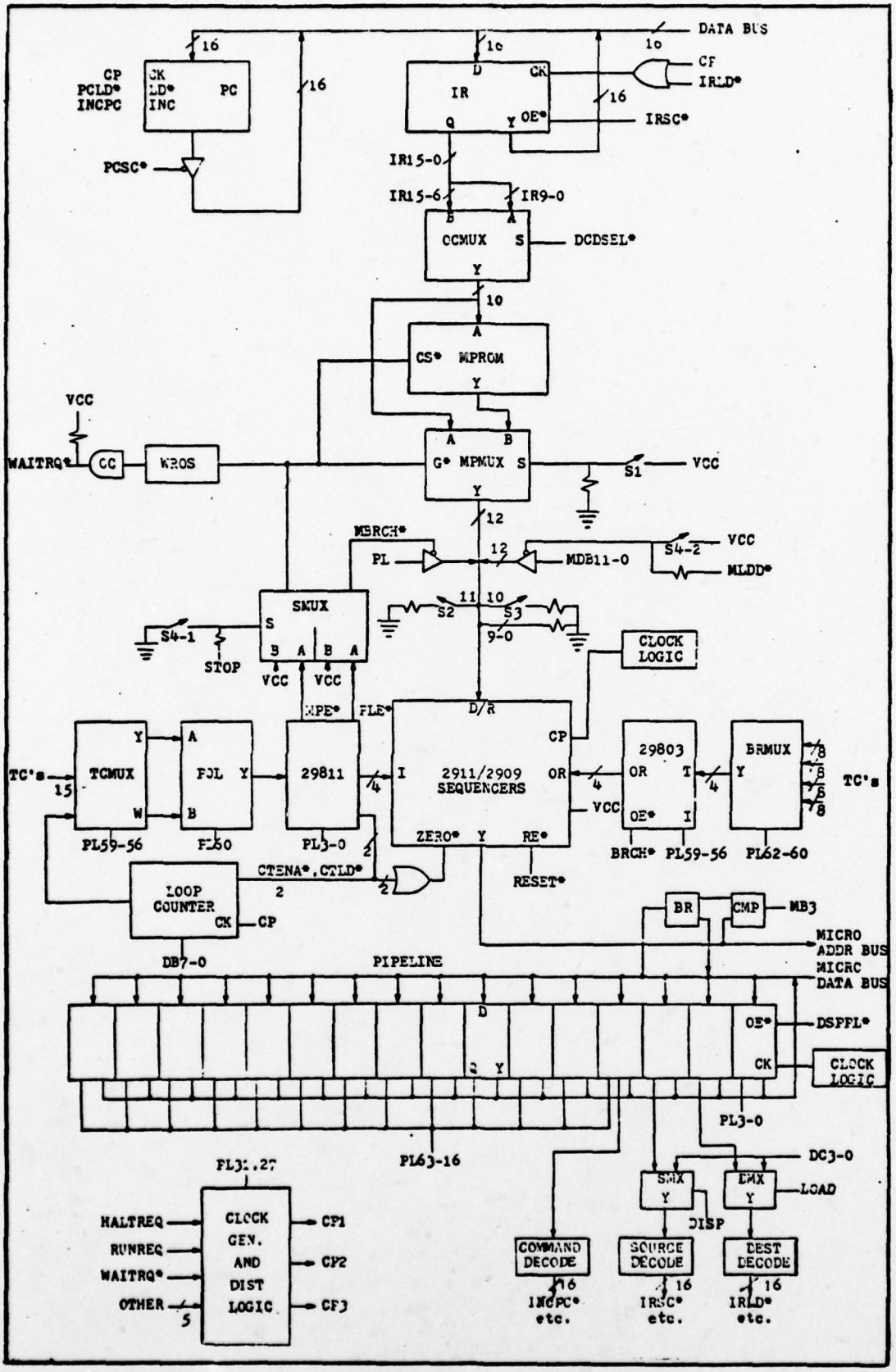


Figure A8. CCU Block Diagram

as well as control signal decoders and master clock generation/ distribution logic. CCU operation will be explained by examining typical transfers occurring chronologically during the fetch and decode of a single machine instruction.

Initially, the PC will be either incremented by INCPC or loaded from the DB by PCLD*. This provides the address of the next machine instruction. Control signal PCSC* will then be used to gate this address onto the DB for transfer to the memory. A memory read cycle yields the next instruction which is loaded from the DB into the IR on the next low-to-high clock transition when IRLD* is low. As controlled by DCDSEL*, OCMUX selects the op code portion of the IR and applies it to the address lines of the Mapping PROM (MPROM). MPROM contains a unique microroutine starting address for each particular op code, thus allowing flexible CS organization. S1 controls MPMUX which allows the user to bypass MPROM and use the op code directly as the microroutine starting address.

The MPMUX tri-state outputs are one of the three starting address sources. The other two sources as PL bits 47-36 and DB bits 11-0. The PL bits are used during micro-branching operations and the DB bits are used to load a starting address from the front panel.

Regardless of the source, the starting address is loaded into the Am2911/Am2909 microsequencer D/R inputs, routed to the Y output, and then gated onto the MAB. The corresponding microword is fetched from CS and loaded into the PL on the next low-to-high clock transition. These PL bits are used

as control signals throughout MIME. Three of the four-bit PL fields are decoded to provide 16 control signals each. To provide front panel load and display capabilities, the BUS SRC and BUS DST fields are routed through multiplexers SMX and DMS respectively and then decoded.

Addresses of subsequent microwords to be fetched from CS are generated by the microsequencers as controlled by PL bits 3-0. The Am2911/Am2909 architecture allows micro level straight line programming as well as conditional branching, looping, and subroutining. Which of these that occurs is controlled by 1 of 16 test conditions selected by the TCMUX and POLMUX. The selected test condition and PL bits 3-0 are inputs to the AM29811 which in turn controls the microsequencers, MPROM starting address source, and Micro Loop Counter (MLC).

When loaded with a value N, the MLC allows N+1 executions of a micro level loop. The MLC is an 8-bit counter, thus the maximum repetition of a loop is 256. Reference 2 contains a detailed discussion of the Am29811 as well as the Am29803 discussed below (Ref 2:1-9 to 1-15, 2-11 to 2-21).

The Am29803 provides a method of simultaneously testing up to four condition codes selected by BRMUX and then branching to the microaddress specified by the test result. This is accomplished using the branch table technique discussed in section III of this manual.

Also included in the CCU is clock generation and distribution logic. This logic generates the master clock CP1 and

two ALU clocks, CP2 and CP3. The latter two are controlled by PL or front panel control signals that specify the desired ALU operation.

Several features have been provided specifically for checkout and troubleshooting purposes. As previously discussed, S1 allows the user to bypass the MPROM during checkout of new microroutines or during module tests. Switch S4 inhibits front panel loading of starting addresses in order to allow the complete Am29811/29803 capabilities to be used for troubleshooting. Switches S2 and S3 allow the lowest address of the selected 1K block of CS to be used as the starting address when MIME is initialized or manually reset.

Control Store. Figure A9 depicts the CS1 module which contains 1K microwords of EPROM and 256 microwords of RAM. Figure A10 depicts the CS2 module which contains provision for 3K microwords of EPROM. As in the memory module, CS read and write cycles will be discussed.

In the run mode, once CS read operation is accomplished each clock cycle. The address generated by the CCU is decoded to enable one block of RAM or EPROM. The addressed word of the enabled block is routed through the Control Store Buffer (CSB) onto the MDB to be clocked into the PL. A toggle switch is provided on the CS1 module to allow the user to select between EPROM and RAM for the lowest 256 microwords with the switch in the RAM position toward the front panel, addresses 000-0FF use RAM, while all others use EPROM. With the switch in the PROM position, all addresses use EPROM.

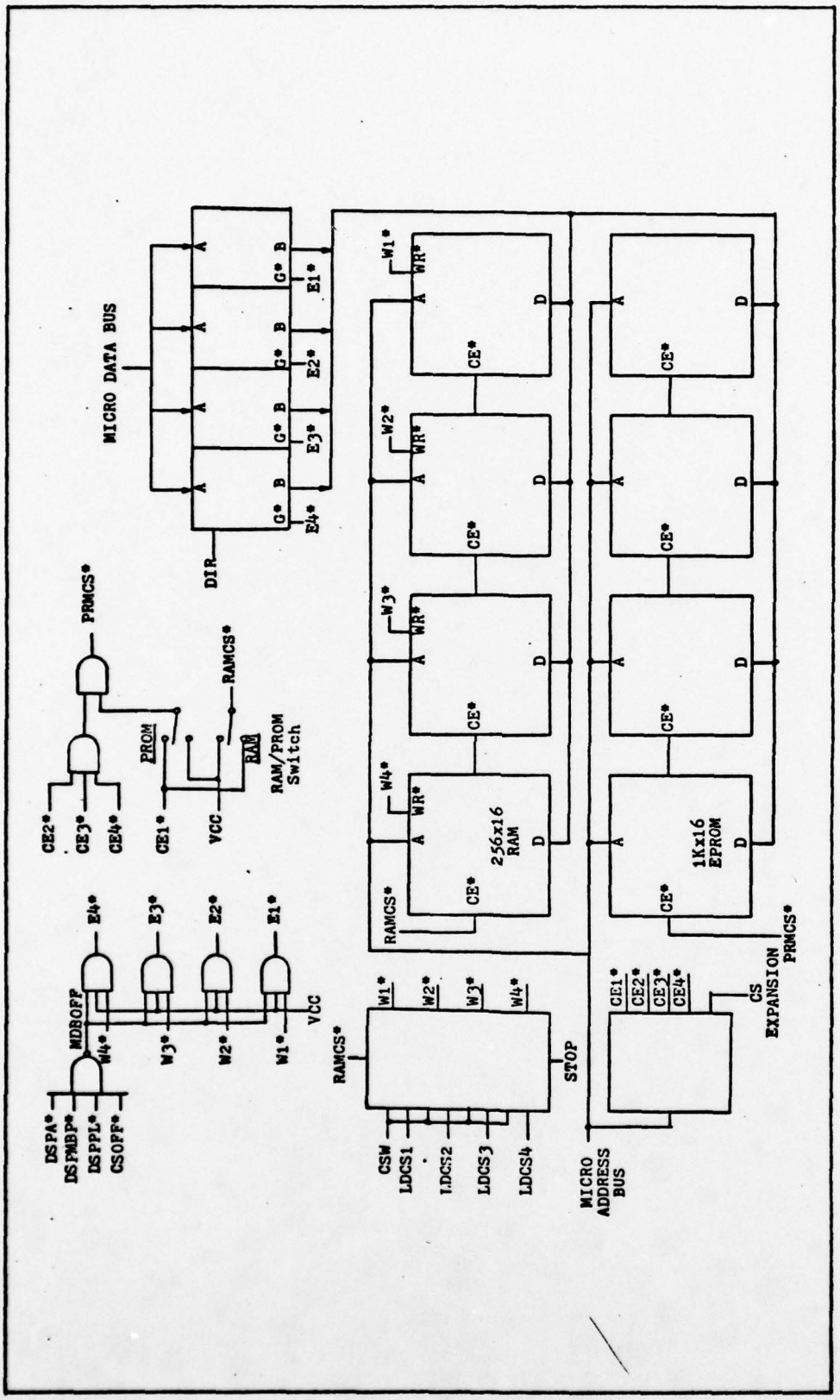


Figure A9. CS1 Block Diagram

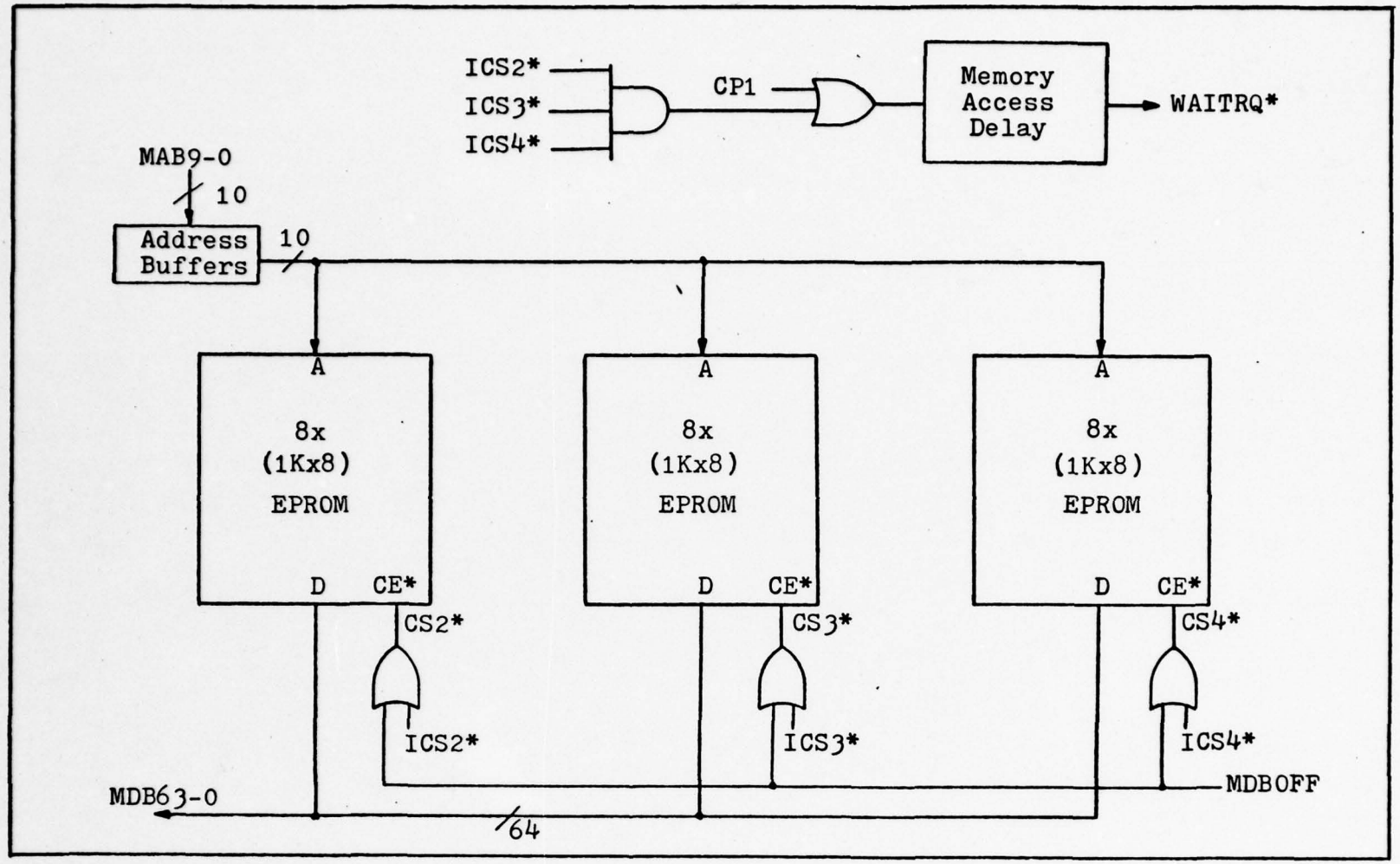


Figure A10. CS2 Block Diagram

CS is not presently writable in the run mode. Control line CSW may be used to write to CS under program control providing appropriate hardware buffers are added between data source and MDB.

Data may be written to the CS RAM in the stop mode by selecting CS and using LOAD or LOAD NEXT. Signal MDBOFF is low with CS selected. Depressing LOAD (NEXT) causes MDBOFF to go high, allowing CS portions selected by front panel FLDEN switches (Wi*low) to be written into. MDBOFF inhibits non-selected CS portions (Wi*high) by disabling corresponding CSB portions.

CS data is displayed from the front panel by depressing DISP or DISP NEXT. This latches the CS output into the micro display. Selecting PL, A, or MBP on the front panel causes MDBOFF to go high, disabling the CSB and isoliating CS1 from the MDB.

I/O. The I/O module of Figure A11 contains the Am2914 Vectored Priority Interrupt Controller, the BAR, the WCR, I/O TC MUX, Direction flip-flop (DFF), the I/O Buffer Register (IOBR), and a serial I/O port.

The Am2914 (Ref 4) is used to control 16 levels of vectored priority interrupt I/O operations. Interrupt requests are accepted on 16 lines, and signal IRQ* is pulled low if the highest priority non-masked request is greater than or equal to SR contents. The sixteen AM2914 instructions are summarized in the microprogramming section of this manual. Both the SR and MR may be loaded under program control with these

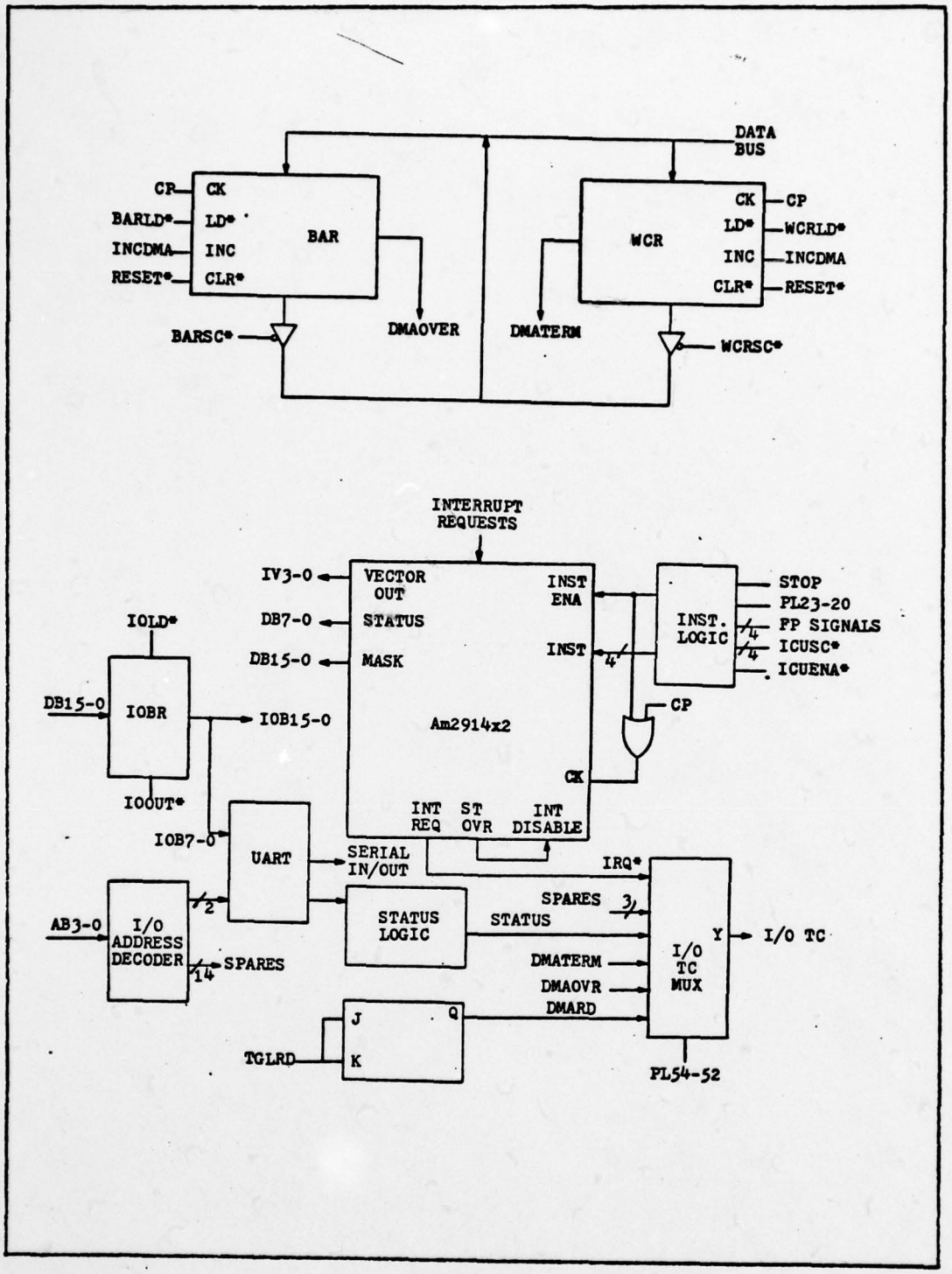


Figure A11. I/O Block Diagram

instructions, or from the front panel. The instruction logic interfaces the Am2914 to the front panel and prevents inadvertent contention on the DB.

During DMA operation, the BAR is loaded with the initial memory address while the WCR is loaded with the two's complement of the number of words to be transferred. The BAR is gated onto the AB for each DMA cycle, and the BAR and WCR are incremented. The DMATERM flag is used to indicate completion of the desired number of transfers. The DMAOVR flag is set if the memory address overflows.

The I/O TC MUX selects 1 of 8 I/O related condition codes for testing the CCU. One of these codes is DMARD, the DFF output. DFF is loaded with the transfer direction during DMA initialization and is tested to determine the direction of each subsequent DMA transfer.

The I/O module also contains the I/O Buffer Register (IOBR), a bidirectional register which communicates between the DB and the I/O Bus (IOB). Address decoding for up to 16 I/O ports is also provided by decoding the 4 least significant bits of the address bus. Two of these ports are used to service an RS-232 serial I/O port which operates at user-selectable rates from 110 to 19200 baud. During an output operation, the IOBR is loaded with the data to be output from MIME. The desired port is addressed by gating the MAR or BAR onto the AB, and the port STATUS is tested. When the STATUS flag is true, the port is ready for data, and the IOWT command is generated to gate the data from the IOBR onto the IOB. An input operation

follows the same steps in reverse, using IORD to latch IOB data into the IOBR.

AUXILIARY. The architecture of MIME has provisions for up to 16 User Definable Auxiliary Functions (UDAF) - UDAFF) which can be registers, flags, timers, etc. The contents of user defined registers are gated onto or off the DB by selecting AUX as the bus source (AUXRD*) or destination (AUXLD*). The individual functions are selected using the UDAF field of the microword. There is also provision for up to 16 User Definable Auxiliary Commands (UDAC) - UDACF) using the UDAC field of the microword. These commands are enabled by the AUXENA* command. Four User Defined Auxiliary Test Conditions (UDTCB - UDTCE) are provided to the CCU TC MUX. Finally, system clock and several interrupts are provided.

All of the above mentioned signals are wired into an edge connector at location I of the card cage. Definitions of the pins of this connector are in Table AIII. By properly designing a circuit card to plug into this connector, the user can add various hardware capabilities to MIME.

CHASSIS DETAILS. The chassis was designed to provide maximum access to MIME components. The front panel is hinged (restricted to 130° travel) to allow access to FP1 wiring. Removal of the plexiglass faceplate allows access to all FP1 components. The remaining circuit cards are rack mounted, and access to components and wiring may be obtained by card removal. An extender card has been provided to allow access to one card under operating conditions. Access to the

Table AIII

AUX Edge Connector Signal Definitions

Pin	Signal	Pin	Signal
IE001	VCC1	IE055	DB04
IE010	UDAF0*	IE056	DB06
IE011	UDAF1*	IE057	DB08
IE012	UDAF2*	IE059	DB10
IE013	UDAF3*	IE060	DB12
IE014	UDAF4*	IE061	DB14
IE015	UDAF5*	IE062	GND1
IE016	UDAF6*	IE070	UDAF8*
IE018	UDAF7*	IE071	UDAF9*
IE019	AUXRD*	IE072	UDAFA*
IE020	UDTCB	IE073	UDAFB*
IE021	UDTCC	IE074	UDAFC*
IE022	CP1A	IE075	UDAFD*
IE023	UDTCD	IE076	UDAFE*
IE024	UDTCE	IE077	UDAFF*
IE025	CCRCLK	IE084	AUXLD*
IE026	IC	IE090	P09
IE027	IZ	IE093	P07
IE028	IN	IE095	GND2
IE029	P08	IE096	P10
IE033	P06	IE097	P12
IE034	VCC2	IE098	P14
IE035	P02	IE104	GND3
IE036	P11	IE105	UDAC8*
IE037	P13	IE106	UDAC9*
IE038	P15	IE107	UDACA*
IE041	P03	IE108	UDACB*
IE043	VCC3	IE109	UDACC*
IE044	UDAC0*	IE110	UDACD*
IE045	UDAC1*	IE111	UDACE*
IE046	UDAC2*	IE112	UDACF*
IE047	UDAC3*	IE113	GND4
IE048	UDAC4*	IE114	DB01
IE049	UDAC5*	IE115	DB03
IE050	UDAC6*	IE116	DB05
IE051	UDAC7*	IE117	DB07
IE052	VCC4	IE118	DB09
IE053	DB00	IE120	DB11
IE054	DB02	IE121	DB13
		IE122	DB15

backplane has been provided by a removable panel on the bottom of the MIME cabinet. All voltages required by the MIME circuitry are provided by chassis-mounted regulated power supplies.

For user convenience, a Miscellaneous Signal Panel (MSP) has been provided which contains a HEX/OCTAL select switch (macro display), power connections and test points. The HEX/OCTAL switch determines whether data displayed on the macro level is in hexadecimal or octal format. The power connections may be used as a power source for external experimental setups. Several test points were provided to allow easy access to dynamic signals of interest.

II. Operation

This section presents procedures used to load and execute macro and micro level programs using the MIME front panel. It is assumed that the user is familiar with the System Description section of this manual. Capitalized words refer to FP1 switches and indicators.

Macroprogram Loading

Machine language programs are loaded into MEM from FP1 in the following manner:

1. Select Hex/Octal format (on MSP)
2. Select MAR (macro level register select)
3. LOAD MAR with initial memory address (select macro level and enter value via KYBRD and LOAD)
4. Select MBR (macro level register select)
5. LOAD (NEXT) memory location with data

Microprogram Loading.

CS contains both read-only and read/write memory for microinstruction storage. Read-only CS can be changed by removing and reprogramming the CS EPROM. Read/write CS can be loaded from FP1 using the following procedure:

1. Place switch on CS1 to RAM position
2. Select A (micro level register select)
3. Load initial microaddress (select micro level and enter value via KYBRD and LOAD)
4. Select CS (micro level register select)

5. LOAD (NEXT) CS location with data

Program Execution

The following procedure is used to execute MIME programs:

1. Place CCU switches 1, 2, and 3 in desired positions (select CS blck, see CCU Module Description)
2. REST MIME to initialize microaddresses
3. LOAD PC with address of first machine instruction (macro level register select enter value via KYBRD and LOAD)
4. Select macro level register for auto display (STEP, AUTO STEP, MSTEP, or AUTO MSTEP modes)
5. Select micro level register for auto display (MSTEP and AUTO STEP modes)
6. Begin program execution in desired mode by use of one of the following:
   a. RUN
   b. STEP
   c. AUTO STEP
   d. MSTEP, RUN, MSTEP
   e. AUTO MSTEP, RUN

   NOTE: For AUTO STEP and AUTO MSTEP, select desired AUTO MSTEP clock rate.
7. Program execution may be halted under program control or by activating one of the following:
   a. HALT
   b. PAUSE
   c. RESET

III. Microprogramming

This section presents information which enables the reader to implement machine instructions as microinstruction modules. The microword fields (related groups of bits) are defined first, and examples of their use are then provided.

Microword Fields

More than 150 signals are used to control MIME. A 150 bit microword would have been extremely flexible but prohibitively expensive when realized in hardware. Therefore, the techniques of encoding and formatting were used to reduce the microword length to a more reasonable 64 bits. As shown in Figure A12, fields 2, 3, and 4 are encoded four bit fields which each provide 16 control signals through use of 1 of 16 decoders. The majority of the other fields are formatted such that they each control two or more functions as specified by a set of steering bits. Each field is discussed in detail below, and summarized in Table AIV.

MICROADDRESS SEQUENCER. This field controls microaddress sequencing using the Am29811 next address control unit in conjections with Am2909/11 microinstruction sequencers (Ref 2: 1.9-2.21). One of the sixteen Am29811 instructions shown in Table AIV must be specified in each microinstruction.

BUS DESTINATION. This field is decoded to control gating of DB information into 1 of 10 registers. As shown in Table AIV, it also control MEM read and write cycles.

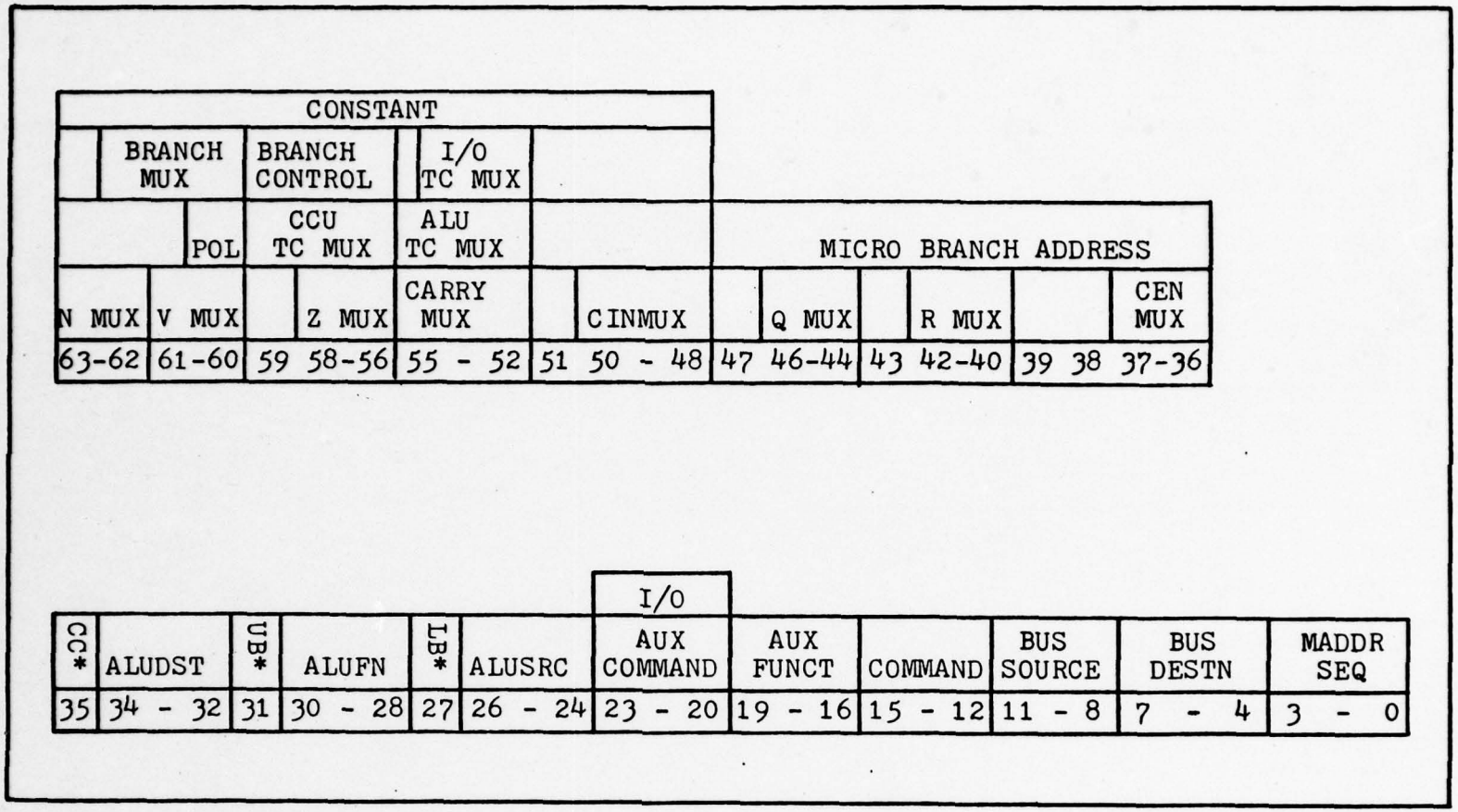


Figure A12. Microword Format

Table AIV

Microword Fields

BUSS DESTINATION BITS 7-4	
CODE	FUNCTION
0	Load PC from DB
1	Load MAR from DB
2	Load MBR from DB
3	Load IR from DB
4	Read from MEM to MBR
5	Write to MEM from MBR
6	Load DBR from DB
7	Load CCR from DB
8	FP load of MR (DO NOT USE)
9	FP load of SR (DO NOT USE)
A	Load BAR from DB
B	Load WCR from DB
C	Load FP display from DB
D	Load AUX from DB
E	Load IOBR from DB
F	No Operation

BUSS SOURCE BITS 11-8	
CODE	FUNCTION
0	Gate PC onto DB
1	Gate MAR onto AB
2	Gate MBR onto DB
3	Gate IR onto DB
4	Halt Command
5	Gate AUX onto DB
6	Gate ALU Output onto DB
7	Gate CCR onto DB
8	FP display of MR (DO NOT USE)
9	Enable ICU operations using DB
A	Gate BAR onto AB
B	Gate WCR onto DB
C	Gate FP data onto DB
D	Gate CONSTANT onto DB
E	Gate IOBR onto DB
F	No Operation

MICROADDRESS SEQUENCER BITS 3-0						
CODE	INSTRUCTION	TEST INPUT	NEXT ADDR SRC.	FILE	CNTR	ENA (1)
0	Jump to address zero	X	D	HOLD	----	PL
1	Conditional jump-to-subroutine; PL address	L H	MPC D	HOLD PUSH	HOLD HOLD	PL PL
2	Jump to Mapping PROM output	X	D	HOLD	HOLD	PROM
3	Conditional jump to PL	L H	MPC D	HOLD HOLD	HOLD HOLD	PL PL
4	Push file; conditionally load counter	L H	MPC MPC	PUSH PUSH	HOLD LOAD	PL PL
5	Jump to subroutine; address conditionally R or PL	L H	R D	PUSH PUSH	HOLD HOLD	PL PL
6	Conditional jump to initial microaddress (CCU S2 and S3)	L H	MPC D	HOLD HOLD	HOLD HOLD	N.A. N.A.
7	Jump to address; conditionally R or PL	L H	R D	HOLD HOLD	HOLD HOLD	PL PL
8	Repeat loop if counter ≠ 0 (must test CT*)	L H	FILE MPC	HOLD POP	DEC HOLD	PL PL
9	Repeat PL if counter ≠ 0 (must test CT*)	L H	D MPC	HOLD HOLD	DEC HOLD	PL PL
A	Conditional return from subroutine	L H	MPC FILE	HOLD POP	HOLD HOLD	PL PL
B	Conditional jump to PL and POP file	L H	MPC D	HOLD POP	HOLD HOLD	PL PL
C	Load counter and continue	X	MPC	HOLD	LOAD	PL
D	Repeat loop if condition ≠ 1	L H	FILE MPC	HOLD POP	HOLD HOLD	PL PL
E	Continue to next addr.	X	MPC	HOLD	HOLD	PL
F	Jump to PL address	X	D	HOLD	HOLD	PL

(1) NOTES: ENA gates specified data to 2909/11 D/R inputs. D may be selected as next address source on same clock cycle. R is loaded on clock rising edge and may be selected on next clock cycle. MPC contains current address + 1.
PL = microbranch address field of current microword.

Table AIV (Continued)

INPUT/OUTPUT FIELD

INTERRUPT CONTROL UNIT BITS 23-20	
CODE	FUNCTION
0	Master clear: clear all interrupts, MR, SR; enable interrupt requests
1	Clear all interrupts
2	Clear interrupts selectively from DB data
3	Clear interrupts selectively from MR data(requires code 9 in BUSS SOURCE field)
4	Clear interrupt associated with last vector read
5	Read interrupt vector (IV3-0); load SR+1 into SR (enables 16-way branch)
6	Gate SR onto DB (requires code 9 in BUSS SOURCE field)
7	Gate MR onto DB (requires code 9 in BUSS SOURCE field)
8	Set MR (inhibits all interrupts)
9	Load SR from DB
A	Bit clear MR from DB
B	Bit set MR from DB
C	Clear MR (enables all interrupts)
D	Disable interrupt request (IRQ*)
E	Load MR from DB
F	Enable interrupt request (IRQ*)

COMMAND BITS 15-12	
CODE	FUNCTION
0	Increment PC
1	Increment MAR
2	Increment WCR & BAR
3	Toggle READ FF
4	Enable 16-way Branch
5	Reset Memory FF
6	Load A address latch from DB
7	Load B address latch from DB
8	Load A and B latches from DB
9	Decode lower part of IR
A	Increment B latch
B	Decrement B latch
C	Enable Am 2914 (ICU) commands
D	Write to I/O device
E	Read from I/O device
F	Enable AUX COMMAND

Table AIV (Continued)

ALU PROCESSOR FIELDS

CC* BIT 35	
CODE	FUNCTION
0	Load CCR
1	Load MCCR

ALU DEST BITS 34-32		
CODE	Y	RAM/Q STORE
0	F	Q := F
1	F	no storage
2	(A)	B := F
3	F	B := F
4	F	B := F/2, Q := Q/2
5	F	B := F/2
6	F	B := 2F, Q := 2Q
7	F	B := 2F

UB* BIT 31	
CODE	FUNCTION
0	Enable upper byte store, ALU and MEM
1	Disable upper byte store, ALU and MEM

ALU FN BITS 30-28	
CODE	ALU FUNCTION
0	$R + S + C_{in}$
1	$S + R^* + C_{in}$
2	$R + S^* + C_{in}$
3	R ∨ S
4	R ∧ S
5	$R^* \wedge S$
6	R ∨ S
7	$(R \vee S)^*$

LB* BIT 27	
CODE	FUNCTION
0	Enable lower byte store, ALU and MEM
1	Disable lower byte store, ALU and MEM

ALU SRC BITS 26-24		
CODE	SOURCE OPERANDS	
	R	S
0	A reg	Q reg
1	A reg	B reg
2	∅	Q reg
3	∅	B reg
4	∅	A reg
5	DBR	A reg
6	DBR	Q reg
7	DBR	∅

SHIFTING LINKAGE FIELDS

QMUX BITS 46-44		
	LEFT SHIFT	RIGHT SHIFT
CODE	Q∅	Q15
0	RAM15	RAM∅
1	Q15	Q∅
2	CARRY	CARRY
3	GND	GND
4	VCC	VCC
5	spare	spare
6	spare	DBR15
7	spare	RAM8

RMUX BITS 42-40		
	LEFT SHIFT	RIGHT SHIFT
CODE	RAM∅	RAM15
0	RAM15	RAM∅
1	Q15	Q∅
2	CARRY	CARRY
3	GND	GND
4	VCC	VCC
5	Q7	spare
6	spare	DBR15
7	RAM7	RAM8

CENMUX BITS 37-36		
	LEFT SHIFT	RIGHT SHIFT
CODE	RAM8	RAM7
0	GND	GND
1	CARRY	CARRY
2	spare	DBR7
3	RAM7	RAM8

Table AIV (Continued)

ALU CONDITION CODE FIELDS, CARRY IN FIELD

NEG MUX BITS 63-62	
CODE	NEG SOURCE
0	NEG (repeat)
1	Upper/Lower Byte Neg
2	GND (clear)
3	VCC (set)

OVER MUX BITS 61-60	
CODE	OVER SOURCE
0	OVER (repeat)
1	Upper/Lower Byte Over
2	GND (clear)
3	VCC (set)

ZERO MUX BITS 58-56	
CODE	ZERO SOURCE
0	spare
1	spare
2	Lower Byte Zero
3	Upper Byte Zero
4	ZERO (Repeat)
5	Word Zero
6	GND (Clear)
7	VCC (Set)

CARRY MUX BITS 55-52	
CODE	CARRY SOURCE
0	CARRY (Repeat)
1	CARRY* (Invert)
2	spare
3	spare
4	spare
5	spare
6	AC1 (Aux carry)
7	Lower Byte Carry
8	Upper Byte Carry
9	RAMØ
A	RAM15
B	Q15
C	RAM7
D	RAM8
E	GND (Clear)
F	VCC (Set)

NOTE: Codes 9-D require a shifting storage operation to be specified as ALU DEST.

CIN MUX BITS 50-48	
CODE	CARRY IN
0	spare
1	spare
2	Micro CARRY
3	Micro CARRY*
4	CARRY
5	CARRY*
6	GND (clear)
7	VCC (set)

Table AIV (Continued)

TEST CONDITION FIELDS

CCU TC MUX BITS 59-56	
CODE	CONDITION CODE
0	BP1 encountered
1	BP2 encountered
2	Use I/O TC field
3	Use ALU TC field
4	MARØ
5	MEM (memory FF)
6	CT* (loop cntr)
7	IR4
8	IR5
9	IR7
A	IR15
B	User Defined
C	User Defined
D	User Defined
E	User Defined
F	VCC (set)

ALU TC MUX BITS 55-52	
CODE	TEST CONDITION
0	spare
1	MFØ
2	MALTB
3	MAGTB
4	MCRY
5	MOVR
6	MZERO
7	MNEG
8	spare
9	FØ
A	ALTB
B	AGTB
C	CARRY
D	OVER
E	ZERO
F	NEG

I/O TC MUX BITS 54-52	
CODE	TEST CONDITION
0	DMAOVR
1	DMATERM
2	DMARD
3	STATUS
4	spare
5	spare
6	spare
7	IRQ*

POL BIT 60	
CODE	POLARITY
0	Positive
1	Negative

16 WAY BRANCH CONTROL FIELDS

BRANCH MUX BITS 62-60	
CODE	TEST CONDITIONS
0	IR3,2,1,0
1	IR6,5,4,3
2	IR9,8,7,6
3	IR12,11,10,9
4	IR15,14,13,12
5	IV3,2,1,0
6	CCR (Z,N,C,V)
7	spare

BRANCH CONTROL BITS 59-56	
CODE	TEST(S) MADE
0	No test
1	t0
2	t1
3	t1,t0
4	t2
5	t2,t0
6	t2,t1
7	t2,t1,t0
8	t3
9	t3,t0
A	t3,t1
B	t3,t1,t0
C	t3,t2
D	t3,t2,t0
E	t3,t2,t1
F	t3,t2,t1,t0

BUS SOURCE. The bus source field, also shown in Table AIV, gates data onto the DB from 1 of 9 registers, and onto the AB from 1 of 2 registers. Both the bus source and destination fields must be specified for each microinstruction. In addition, memory read/write cycles require either the MAR or BAR as bus source to gate addresses onto the AB. The ICU must be the selected bus source during specific Am2914 instructions discussed in the interrupt control field section.

COMMAND. The command field generates 1 of 16 miscellaneous control signals as summarized in Table AIV. This field must be specified for each microinstruction.

AUX COMMAND and AUX FUNCTION. These two fields are user definable. They are each four bits wide, allowing up to 16 operations to be selected by each field. The AUX COMMAND field is enabled by the AUXENA* signal generated from the COMMAND field. AUX FUNCTION is enabled by either AUXRD* or AUXLD* generated by BUS SOURCE or BUS DESTINATION, respectively.

INPUT/OUTPUT. This field uses the instructions shown in Table AIV to control the Am2914 Vectored Priority Interrupt Controller (Ref 4). This field must be used in conjunction with command ICU. In addition, the ICU must be the bus source when reading the SR or MR or clearing interrupts from the MR to avoid multiple DB sources.

ALU SOURCE, FUNCTION, DESTINATION. As shown in Table AIV, these three fields control the ALU Am2901's. ALU source selects 1 of 8 operand pairs; function selects 1 of 8 operations; and destination controls shifting and destination of

processing results.

UB* and LB*. These two 1 bit fields control upper and lower byte ALU and MEM storage respectively. As shown in Table AIV, ALU storage is enabled by selecting the appropriate ALU destination field with UB* and/or LB* low. MEM write is accomplished by selecting a memory write in the BUS DEST field, an AB source in the BUS SOURCE field, and UB* and/or LB* low. ALU storage may be disabled during MEM write with ALU destination OF. UB* and LB* must be specified for each microinstruction.

CC*. CC* determines whether the condition codes generated as the result of arithmetic operations are latched by the CCR (CC* low) or the MCCR (CC*high) as shown in Table AIV.

MICROBRANCH ADDRESS. As shown in Figure A12, the Microbranch Address field is also formatted. The contents of this field conditionally specifies the address of the next microinstruction when the Microaddress Sequencer field contains 1, 3, 5, 7, 9, B or F.

CENMUX, RMUX, QMUX. These fields are used with the ALU destination field to select appropriate shifting linkages as shown in Table AIV.

I/O TC, ALU TC, CCU TC, POL. As shown in Table AIV, these fields select condition codes for testing the CCU. I/O TC selects 1 of 8 condition codes generated in the I/O module, while ALU TC selects 1 of 16 generated in the ALU. Eight of the ALU condition codes are stored in the CCR and eight in the MCCR. CCU TC selects 1 of 16 condition codes,

including those preselected by the I/O TC and ALU TC. POL specifies positive or negative test logic.

CARRY, ZERO, OVR, NEG. These fields select sources for ALU condition codes CARRY, ZERO, OVR, and NEG. As shown in Table AIV, UB* is used with OVR and NEG to automatically select the correct byte when the ALU upper/lower byte action (code 1) is selected.

CIN. CIN specifies the ALU carry in during word, upper byte, and lower byte operations. CIN is shown in Table AIV and must be specified for word or byte arithmetic operations.

BRANCH CONTROL and BRANCH MUX. These two fields shown in Table AIV are used to control the Am29803 16-Way Branch Control Unit (Ref 2:1.13-2.15). The branch control field specifies simultaneous testing of up to four condition codes selected from one of eight sets of four by the branch MUX. The Am29803 output is OR'ed with the four least significant next microaddress bits, resulting in a branch to 1 of 16 sequential locations in CS. Each of the 16 locations of this "branch table" could contain an additional branch instruction, resulting in 16-way branching capability.

CONSTANT. This 16 bit field is gated onto the DB when the bus source field contains code D, where it is available for use as a constant, ALU RAM A/B address, interrupt MASK, or to load the Micro Loop Counter.

ALU Operation.

The possible ALU functions, data sources and result destinations have been listed in Table AIV. This section

clarifies how to do arithmetic and logical operations with the ALU.

ALU Sources. The possible operand pairs for an operation with the ALU are listed in Table AIV. A reg and B reg refer to individual registers of the sixteen addressable registers in the Am2901, i.e. R0 through R15 on the front panel. The A and B latches are used to select these registers as ALU operands. Q reg refers to the Q register on the Am2901. Finally, the DBR provides external data into the data inputs of the Am2901. Thus, data on the DB destined for one of the addressable registers must first be loaded into the DBR. The contents of the DBR may then be passed to any of the other registers of the ALU by performing a logical OR between the DBR and 0, and storing the result in the appropriate register.

The A and B Latches. The A and B latches are used to designate which of the Am2901 registers are presented to the ALU inputs/output as operands or result destination. The latches each hold a four bit address allowing selection of any of the sixteen general purpose registers; A and B latches may address the same registers simultaneously.

The A and B latches are loaded from the DB; the A latch is loaded from DB 3-0 and the B latch is loaded from DB 7-4. Consequently, the latches are loadable from any register that can be a bus source, including the output of the ALU. In addition, the contents of the B latch may be incremented or decremented using appropriate commands. This feature is useful when doing multiple precision operations. The commands

used to manipulate the latches are summarized in Table AIV.

Byte Operations. The MIME allows autonomous upper or lower byte operations by disabling storage of the result of the byte not selected i.e., results generated by the ALU are retained only for the byte selected. Microword bits 31 and 27 enable the upper and lower bytes of the ALU. When either bit is set to 1, the corresponding byte is disabled. For an example of autonomous byte operations, see Figure A13.

Carry In. Arithmetic operations with the ALU require the carry in to be specified. This is done using microword bits 50 - 48 as summarized in Table AIV. Figure A14 gives examples of use of the carry in.

ALU Outputs. The results of an ALU operation may be stored in various ways:

- The result may be stored in one of the Am2901 registers by selecting the register as the B latch address.
- The result may be stored in the Am2901 Q register.
- The result may be made available to the DB by gating the ALU output onto the DB. This allows the ALU output to be gated to any register capable of acting as a DB destination. No storage may be specified to prevent disturbing the Am2901 registers.
- The result may be shifted left or right one place before storage into the Am2901 registers as described under "Shifting operations" below.
- Combinations of the above.

Figure A15 depicts examples of ALU output storage.

```
ADDRESS      MICROCODE                    SOURCE PROGRAM
-------      ---------                    --------------

0000                                      PROGRAM
0000                                      $BYTE OPERATIONS--THE LOWER BYTES OF REGISTERS 2 AND 3 ARE
                                                           LOGICAL ORED, WITH THE RESULT STORED IN THE
                                                           LOWER HALF OF REGISTER 2.
                                          $
0000   0023,FFFF,FFFF,8DFE                    LOAD ABLATCH FROM '0023'$.$LOAD THE A & B LATCHES$
0001   040F,FFF3,B1FF,FFFE                    A OR B = BF$ DISABLEUP$.  $DO THE OPERATION, DISABLING THE
                                                                          UPPER BYTE. $
```

Figure A13. Byte Operations

```
                                          $
0002                                      $USE OF THE CARRY-IN FIELD--THE CODE BELOW PERFORMS THE
                                                           DOUBLE PRECISION ADD R3,R4 <--- R1,R2 + R3,R4
                                                           SINCE ONLY ONE WORD MAY BE ADDED AT
                                                           A TIME, THE CARRY FLAG IS USED TO HOLD
                                                           THE INTER-WORD CARRY.
                                          $
0002   0024,FFFF,FFFF,8DFE                    LOAD ABLATCH FROM '0024'$.$LOAD THE A & B LATCHES$
0003   0486,FFF3,01FF,FFFE                    A +:0 B = BF$ SETCC:UPCARRY$.
                                                        $ADD THE LESS SIG. BYTES & SAVE CARRY.$
0004   0013,FFFF,FFFF,8DFE                    LOAD ABLATCH FROM '0013'$.$LOAD THE A & B LATCHES$
0005   0404,FFF3,01FF,FFFE                    A +:CARRY B = BF$.$ADD THE MOST SIG. BYTES AND ADD IN
                                                                  THE CARRY FROM THE LESS SIG. RESULT.$
0006                                      $

                                          $
0006                                      $USE OF THE CARRY-IN FIELD--INCREMENTING/DECREMENTING THE
                                                           CONTENTS OF A REGISTER.
                                          $
0006   0000,FFFF,FFFF,7DFE                    LOAD BLATCH FROM '0'$.$REGISTER 0 WILL BE USED$
0007   0407,FFF3,03FF,FFFE                    B +:1 0 = BF$.        $INCREMENT R0$
0008   0406,FFF3,13FF,FFFE                    B -:0 0 = BF$.        $DECREMENT R0$
0009                                      $
```

Figure A14. Using the Carry In Field

```
                                        *
0009                                    *STORAGE INTO THE B LATCH ADDRESSED REGISTER*
0009    040F,FFF3,31FF,FFFE                 A OR B = BF).*ASSUMES LATCHES HAVE BEEN ALREADY SET*
000A                                    *
                                         STORAGE INTO THE Q REGISTER*
000A    040F,FFF0,31FF,FFFE                 A OR B = QF).
                                        *
                                         STORAGE INTO THE MBR VIA THE DB*
000B    040F,FFF1,37FF,F62E                 DBR OR 0 = OF) LOAD MBR FROM ALU).
                                              *THIS PASSES THE CONTENTS OF THE DBR TO THE MBR*
000C                                    *
```

Figure A15. ALU Storage Examples

Shifting Operations. As stated above, the ALU outputs may be shifted before storage. There are basically two modes of shifting: shifting the output left or right before storage in B register and shifting as above plus shifting the Q register left or right and restoring the result in Q. The latter is very helpful in double precision operations. These shifts are specified by appropriate code in the ALU DEST field of the microword (bits 34 - 32). When a shift operation is specified, the input(s) to the vacated position(s) of the shifted register(s) must be specified. The input(s) may be chosen to effect logical, arithmetic, and/or cyclic shifts and are controlled by the Q MUX (bits 46 - 44) for the Q register, the R MUX (bits 42 - 40) for the B register and the CEN MUX (bits 37 - 36) for the linkage between bytes of the B register. Examples are given in Figure A16.

Condition Codes. The MIME is provided with duplicate condition code registers designated CCR and MCCR. Each register has bits defined as shown in Figure A17. Either the CCR or MCCR is set during each ALU operation, as determined by bit 35 of the microword. When this bit is high, the MCCR is loaded with the condition resulting from the current ALU operation. When the bit is low, the CCR is loaded. When the CCR is loaded, presence of a logic 1 in the NEG, ZERO, OVR, or CARRY bits will cause an indicator on the front panel to illuminate. The CCR was designed to be used as the macro condition code register, while the MCCR was envisioned to be used by the microprogrammer and is not available at the front

```
ADDRESS      MICROCODE            SOURCE PROGRAM
-------      ---------            --------------

 000D                              PAGE
 000D                             * SHIFTING EXAMPLES-- USING SHIFTED ALU DESTINATIONS, A SINGLE OR
                                                       DOUBLE WORD SHIFT (LEFT OR RIGHT) CAN BE
                                                       PERFORMED USING THE ALU. THESE CAN BE ARITH-
                                                       METIC, LOGICAL OR CIRCULAR SHIFTS BY SPECIFYING
                                                       THE APPROPRIATE BITS TO SHIFT IN. A SINGLE
                                                       WORD RIGHT CIRCULAR SHIFT OF THE B REGISTER
                                                       WOULD BE
                                  *
 000D    040F,F035,33FF,FFFE              B OR 0 = RSBF:RAM:RAM0,CENTER:RAM8!.
                                  *                    NOTE THAT ON SHIFTS OF THE B REGISTER, THE
                                                       WORD IS TREATED AS 2 8-BIT BYTES AND THE
                                                       BITS SHIFTED IN TO EACH BYTE MUST BE
                                                       SPECIFIED. A DOUBLE WORD LOGICAL SHIFT
                                                       OF THE Q AND B REGISTER WOULD BE
                                  *
 000E    040F,0336,33FF,FFFE              B OR 0 = LSQBF:Q:RAM15,RAM:0,CENTER:RAM7!.
                                  *                    IN THIS CASE, THE Q REGISTER WAS THE MOST
                                                       SIGNIFICANT HALF OF THE DOUBLE WORD. AN
                                                       ARITHMETIC RIGHT SHIFT HAS TO DO SIGN
                                                       EXTENSION AND HAS TO USE THE DBR TO SAVE
                                                       THE SIGN FOR SHIFTING IN, AS IN
                                  *
 000F    040F,FFF1,32FF,F66E              Q OR 0 = OF! LOAD DBR FROM ALU!.
 0010    040F,6134,33FF,FFFE              B OR 0 = RSQBF:Q:DBR15,RAM:Q0,CENTER:RAM8!.
```

Figure A16. Shifting Examples

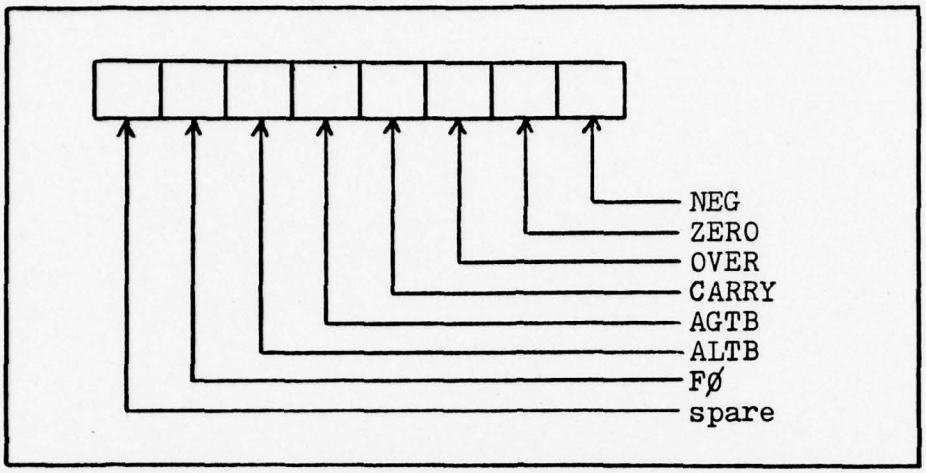


Figure A17. Condition Code Register Format

panel. However, both the CCR and MCCR are available for use as test conditions to construct branches in the microprogram as explained under "Sequencer Operation" below.

The NEG, OVR, ZERO and CARRY bits can take their values from a number of sources as indicated in Table AIV. Each bit is controlled by a separate field in the microword. Note that the MCCR can be loaded from the CCR (NEG, etc.) but not from itself (MNEG, etc.). This means that if the contents of the MCCR are to be preserved, the CCR must be set (possibly from itself) during an ALU operation. Also note that for the RAM0, RAM15, Q15, RAM7 and RAM8 options in the CARRY MUX field to be valid, a shifting operation must be specified in the ALU DEST field. These selections perform as do the QMUX and RMUX options previously described. Figure A18 provides examples of using the CCR and MCCR in microprogramming.

```
ADDRESS      MICROCODE              SOURCE PROGRAM
-------      ---------              --------------

0000                                PROGRAM
0000                                *SET CONDITION CODES EXAMPLES--CONDITION CODES CAN ONLY BE SET IN
                                           THE SAME  MICROINSTRUCTION AS AN ALU OPERATION.
                                           ON ANY ALU OPERATION, CONDITION CODES ARE SET USING THE
                                           VALUE (OR COMMANDS) IN THE CONDITION CODE MUX FIELDS
                                           A 'DO NOTHING' INSTRUCTION SETS THE CCR FROM ITSELF.
                                    *
0000   040F,FFFF,FFFF,FFFE                  CONTINUE!.
                                    *
                                    *
                                    *
0001                                *          IN AN ARITHMETIC OPERATION, THE CONDITION CODES WOULD
                                           MOST NORMALLY BE SET FROM WORD ZERO, BYTE NEG, UPPER BYTE
                                           CARRY, AND OVERFLOW.
                                    *
0001   5586,FFF1,01FF,FFFE          *       A +!0 B = 0F! SETCC!WZERO,BNEG,UPCARRY,BOVER!.
                                    *
                                    *
0002                                *          THE MICRO CONDITION CODES MAY BE SET BY SELECTING
                                           CC* TO A ONE.
                                    *
0002   5586,FFF9,01FF,FFFE          *       A +!0 B = 0F! SETMCC!WZERO,BNEG,UPCARRY,BOVER!.
                                    * *
```

Figure A18. Setting the CCR/MCCR

Register Transfers

This section describes how to program register transfers using the DB. The general rules for loading any register from any register are shown in Figure A19. The Micro Loop Counter and A and B latches may also be loaded from the DB and are included in Figure A19. Note that the ALU may also be a bus source. Examples of register transfer are included throughout this section.

Two registers deserve special attention because their operation is different than that of the other registers. They are the MBR and the IOBR. The design of either register is depicted in Figure A20. Note that data from the DB is loaded into a physically different register than data destined for the DB. Because of this, the sequence of instructions in Figure A21a will not work as might be expected. The data from address 1 will not be loaded into address 2 since the MBR loaded from memory is not the same physical register as the MBR that writes to memory. The sequence of Figure A21b will perform the required action. The sequence of Figure A21c cannot be made to work correctly, since the MBR may not load itself from the memory bus. The IOBR operates in exactly the same manner as the MBR.

Memory Access

This section describes programmed memory access. To fetch data from memory, the following steps must be taken:

- Load the MAR with the address of the data.
- Perform a memory read. This puts the data in the MBR.

BUS SOURCE \ BUS DEST'N	PC	MAR	MBR	IR	AUX	CCR	MR	SR	WCR	BAR	FP	IOBR	DBR	Q	RO-RF	LOOP CNTR.	A/B LATCHES
PC	Y	Y	Y	Y	U	Y	Y	Y	Y	Y	Y	Y	Y	AD	AD	S	L
MAR	N	N	N	N	N	N	N	N	N	N	H	N	N	N	N	N	N
MBR	Y	Y	*	Y	U	Y	Y	Y	Y	Y	Y	Y	Y	AD	AD	S	L
IR	Y	Y	Y	Y	U	Y	Y	Y	Y	Y	Y	Y	Y	AD	AD	S	L
AUX	U	U	U	U	N	U	U	U	U	U	U	U	U	U	U	SU	LU
ALU	YAB	YAB	YAB	YAB	YAB	UAB	YAB	YAB	YAB	YAB	YAB	YAB	YAB	AD	AD	SAB	LAB
CCR	Y	Y	Y	Y	U	Y	Y	Y	Y	Y	Y	Y	Y	AD	AD	S	L
MR	YI	YI	YI	YI	UI	YI	YI	YI	YI	YI	YI	YI	YI	ADI	ADI	SI	N
SR	YI	YI	YI	YI	UI	YI	YI	YI	YI	YI	YI	YI	YI	ADI	ADI	SI	N
WCR	Y	Y	Y	Y	U	Y	Y	Y	Y	Y	Y	Y	Y	AD	AD	S	L
BAR	N	N	N	N	N	N	N	N	N	N	H	N	N	N	N	N	N
FP	Y	Y	Y	Y	U	Y	Y	Y	Y	Y	-	Y	Y	AD	AD	S	L
CONST	YC	YC	YC	YC	UC	YC	YC	YC	YC	YC	YC	YC	YC	ADC	ADC	SC	LC
IOBR	Y	Y	Y	Y	U	Y	Y	Y	Y	Y	Y	*	Y	AD	AD	S	L
DBR	AB	AB	AB	AB	UAB	AB	AB	AB	AB	AB	AB	AB	AB	A	A	ABS	ABL
Q	AB	AB	AB	AB	UAB	AB	AB	AB	AB	AB	AB	AB	AB	A	A	ABS	ABL
RO-RF	AB	AB	AB	AB	UAB	AB	AB	AB	AB	AB	AB	AB	AB	A	A	ABS	ABL

A ALU operation required.

B Bus source is ALU.

C Constant field contains CONST.

D DBR must first be loaded from Bus source; ALU DEST selects destination.

H Hardware only--not possible under microprogram control.

I COMMAND field must enable Am2914.

L Appropriate COMMAND required; BUS SOURCE selects source.

N Not possible.

S Sequencer instruction used; BUS SOURCE selects source.

U User definable.

Y Normal bus transfer; BUS SOURCE selects source; BUS DESTINATION selects destination.

* See text.

Figure A19. Allowable Register Transfers

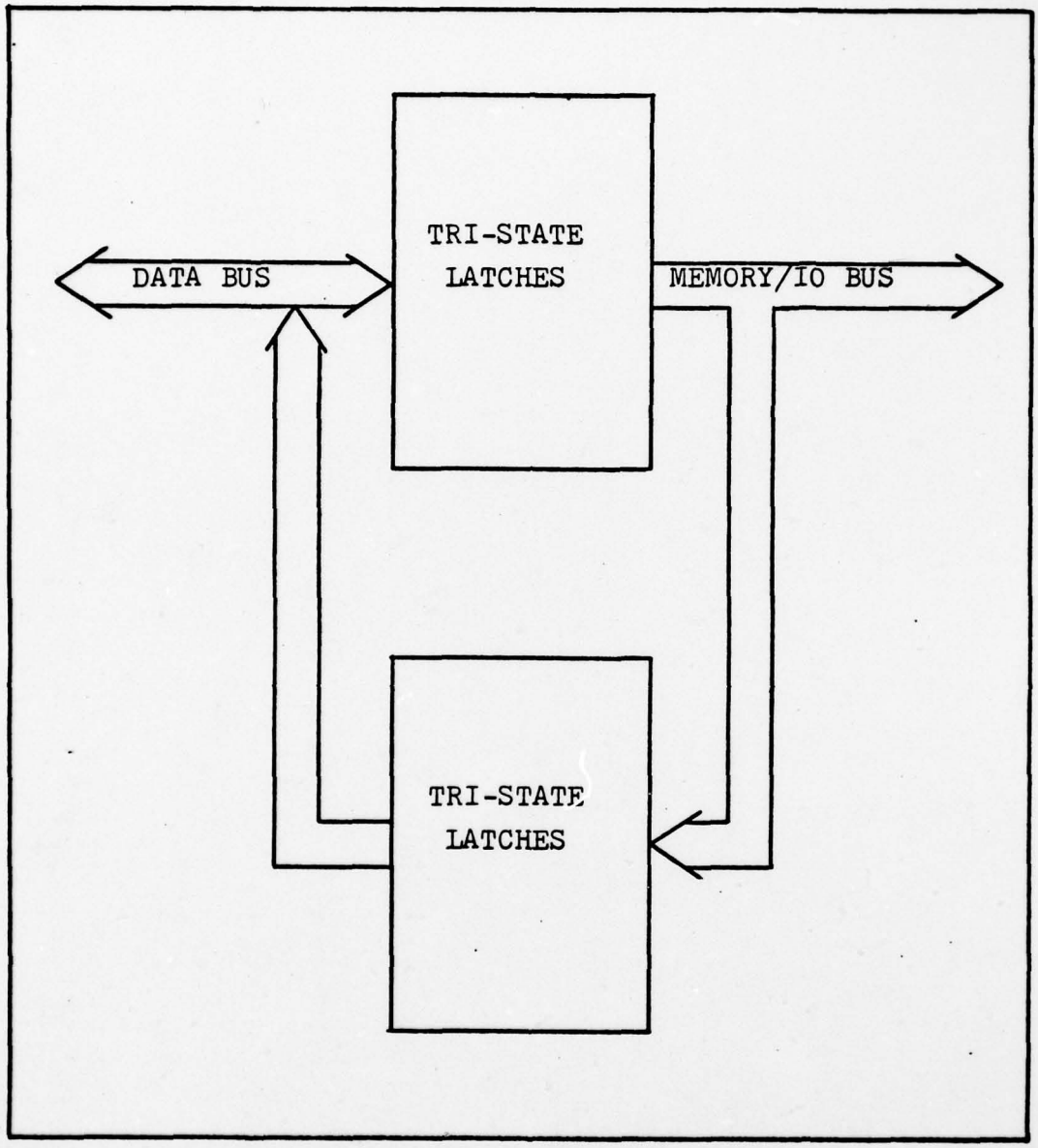


Figure A20. MBR/IOBR Configuration

```
                                    ♦
000C                                ♦THE MBR (IOBR) PROBLEM--ILLUSTATED
                                    ♦
000C   0001,FFFF,FFFF,FD1E               LOAD MAR FROM '0001'♦.♦PREPARE TO ACCESS ADDR 0001♦
000D   040F,FFF9,77FF,F14E               LOAD MBR FROM MEMORY♦.♦DO THE MEMORY READ♦
000E   0002,FFFF,FFFF,FD1E               LOAD MAR FROM '0002'♦.♦PREPARE TO ACCESS ADDR 0002♦
000F   040F,FFF9,77FF,F15E               LOAD MEMORY FROM MBR♦.♦ATTEMPT TO STORE DATA♦
0010                                ♦
```

a

```
                                    ♦
0010                                ♦THE MBR (IOBR) PROBLEM--SOLUTION(PARTIAL)
                                    ♦
0010   0001,FFFF,FFFF,FD1E               LOAD MAR FROM '0001'♦.♦PREPARE TO ACCESS ADDR 0001♦
0011   040F,FFF9,77FF,F14E               LOAD MBR FROM MEMORY♦.♦DO THE MEMORY READ♦
0012   040F,FFFF,FFFF,F22E               LOAD MBR FROM MBR♦.♦I.E. LOAD ONE SIDE FROM THE OTHER♦
0013   0002,FFFF,FFFF,FD1E               LOAD MAR FROM '0002'♦.♦PREPARE TO ACCESS ADDR 0002♦
0014   040F,FFF9,77FF,F15E               LOAD MEMORY FROM MBR♦.♦ATTEMPT TO STORE DATA♦
0015                                ♦
```

b

```
                                    ♦
0015                                ♦THE MBR (IOBR) PROBLEM--THE REMAINING PROBLEM
                                    ♦
0015   040F,FFFF,FFFF,F02E               LOAD MBR FROM PC♦.♦ATTEMPT TO USE MBR AS A TEMPORARY
                                                              STORAGE LOCATION♦
0016   040F,FFFF,FFFF,F26E               LOAD DBR FROM MBR♦.♦THE ABOVE STATEMENT LOADED ONE SIDE
                                                              OF THE MBR. THIS STATEMENT READ THE
                                                              OTHER SIDE-CONSEQUENTLY THE DATA
                                                              FROM THE PC DID NOT REACH THE DBR.♦
```

c

Figure A21. The MBR (IOBR) Problem

- Gate MBR onto DB and load desired register.

Storing data in memory requires the following steps:

- Load MAR with the address desired.
- Load the MBR from the desired register.
- Perform a memory write. This moves data from the MBR to memory.

For either of these sequences, the only subtle step is the actual read/write step. If either byte of the ALU is disabled, the corresponding byte of memory is disabled, hence both bytes must be enabled. But, if the ALU is enabled, the condition code and storage options must be set to avoid destroying the data in registers and/or destroying desired condition codes. Figure A22 illustrates a well-constructed memory read sequence. Care must also be taken if data is to be read from one location and written directly to another location. See the section on register transfers for a discussion of using the MBR for this purpose.

Sequencer Operation

The microaddress sequencer determines the address of the next microinstruction to be executed. Through proper manipulation of the microword sequencer field, the programmer may implement conditional branches, loops, and calls to subroutines as well as sequential execution of microprograms. The following paragraphs describe the usage of the microaddress sequencer field.

Microaddress Sources. Depending upon the execution sequence desired, the user may specify that the next micro-

```
ADDRESS      MICROCODE              SOURCE PROGRAM
-------      ---------              --------------

 0017                               ♦EXAMPLE OF A MEMORY ACCESS♦
 0017    040F,FFFF,FFFF,F01E             LOAD MAR FROM PC♦.♦THE LOCATION TO BE ACCESSED MUST BE
                                                  LOADED INTO THE MAR. IN THIS CASE,
                                                  THE DESIRED ADDRESS IS IN THE PC
                                                  AS IT WOULD BE IN AN INSTRUCTION FETCH♦
 0018    040F,FFF9,77FF,F14E             LOAD MBR FROM MEMORY♦.♦THIS LOADS THE DATA FROM MEMORY
                                                       INTO THE MBR. IT MAY THEN BE GATED
                                                       ONTO THE DB FOR FURTHER USE.♦
```

Figure A22. Memory Access Example

address come from any of several sources. The first of these is the Micro Program Counter (MPC). This counter, contained in the Am2909/2911 address sequencers, always points at the current address plus one. Using the MPC thus causes sequential execution of a program.

The microbranch address field of the PL is a second possible source of the next microaddress. This option may be used to effect program branches, loops and calls to subroutines. A register (R) is always loaded with the contents of the preceding microword's microbranch address field. Conditionally selecting the branch address to be either the PL or R allows two-way branches to be accomplished in one clock cycle. This of course assumes that R was correctly loaded in the instruction preceding the conditional branch instruction.

The output of the Mapping PROM (MAP) provides another set of options. The address of the MAP may be selected to be either the high order or the low order bits of the Instruction Register. Thus, part of the contents of the instruction register (e.g. the opcode portion of an instruction) may be decoded into microroutine start addresses. There is also the option to bypass the MAP and use the contents of the IR directly to enter microroutines.

The last option, the file (F), is used for loops and subroutine return linkages. In a call to a subroutine, the return address is pushed onto F (F is a stack) and upon execution of a return instruction, the return address is

popped off the stack. There are also sequencer instructions that may be used solely to manipulate F. Examples of use of all of these options are discussed below.

Sequential Execution. Selecting microcode E will cause the next instruction executed to simply be the next sequential instruction in the microprogram, irrespective of test conditions.

Conditional Branches. Several types of conditional branch instructions are possible. All test the conditions specified by the microword TCMUX fields:

- Conditional Jump to PL (Sequencer code 3). If the result of the test is "true", the next microinstruction will be at the address specified in the microbranch address field of the same microword. Otherwise, the next sequential microinstruction will be executed. This is analogous to the "Branch on Condition" instructions used by many machine languages.
- Conditional Jump to Initial Microaddress (code 6). If the test result is true, the next microinstruction executed will be the instruction at 000_{16}, 400_{16}, 800_{16}, or $C00_{16}$ depending upon the settings of S2 and S3 on the CCU2 board. Otherwise, the next sequential microinstruction is executed.
- Jump to address, conditionally R or PL (code 7). If the test result is "true", the next microinstruction will be the one designated in the microbranch address field. Otherwise, the next microinstruction address comes from the previous microinstruction's microbranch

address field.

Unconditional Branches. An unconditional branch may be caused by using one of the conditional branch instructions and forcing the test condition high. An easier method, and one which allows more parallelism in the microword, is to use one of the following unconditional branch instructions. These instructions do not use the TC MUX fields, and those fields are available for setting the ALU flags or for inputting a constant. The unconditional branch instructions are:

- Jump to Address Zero (code 0). The next microinstruction to be executed will be the one residing at address 000_{16}.
- Jump to Mapping PROM Output (Code 2). The next microinstruction will be the one designated by the output of the mapping PROM.
- Jump to PL Address (code F). The next microinstruction will be the one designated by the microbranch address field of the current microword.

Looping. The MIME provides capabilities to execute a loop either for a specified number of times or until a specified condition occurs. Various looping possibilities are illustrated in Figure A23.

Subroutines. The sequencer allows conditional subroutine calls and returns, however some caution is in order. The subroutine linkage mechanism is implemented as follows:

- During a jump to subroutine, the current microaddress plus 1 (i.e. the return address) is pushed onto a stack (referred to as the file or F in Table AIV).

```
ADDRESS     MICROCODE           SOURCE PROGRAM
-------     ---------           --------------

 0000                            PAGE
 0000                            * LOOPING EXAMPLES--(1) THIS LOOPING CONSTRUCT USES THE LOOP COUNTER TO
                                                         DETERMINE HOW MANY TIMES TO TRAVERSE THE LOOP AND
                                                         A LABEL AS THE LOOPING DESTINATION. CT* IS TRUE
                                                         WHEN THE LOOP COUNTER IS ZERO.
                                 *
 0000   0403,FFFF,FFFF,FDFC                LOADCNTR FROM '3'; .  *LOOP WILL BE EXECUTED 4 TIMES*
 0001   040F,FFFF,FFFF,FFFE     LOOP/HERE: CONTINUE; .      *   LOOP   *
 0002   040F,FFFF,FFFF,FFFE                CONTINUE; .      *   BODY   *
 0003   060F,001F,FFFF,FFF9                IF NOT CT* THEN LOOP AT LOOP/HERE; .  *LOOP COUNTER IS TESTED.
                                                         IF IT IS NOT ZERO, THEN THE PROGRAM
                                                         BRANCHES TO THE LABEL AND THE LOOP COUNTER
                                                         IS THEN DECREMENTED BY ONE. IF IT IS ZERO,
                                                         THE PROGRAM GOES TO THE NEXT MICROINSTRUCTION.
                                                         *
 0004                            * LOOPING EXAMPLES--(2) ALTERNATIVELY, SOME CONDITION OTHER THAN LOOP
                                                         COUNTER=0 COULD BE USED TO TRANSFER OUT OF THE
                                                         LOOP AND THE LOOP COUNTER NOT USED AT ALL.
                                 *
 0004   040F,FFFF,FFFF,FFFE     LOOP/THERE:    CONTINUE; .
 0005   037F,008F,FFFF,FFF3                IF MNEG THEN GO TO END/LOOP; .  *END OF LOOP BASED ON A
                                                                             MICRO CONDITION CODE*
 0006   040F,FFFF,FFFF,FFFE                CONTINUE; .
 0007   1F0F,004F,FFFF,FFF9                LOOP AT LOOP/THERE; .  *UNCONDITIONALLY LOOP*
 0008   040F,FFFF,FFFF,FFFE     END/LOOP:  CONTINUE; .
                                 * LOOPING EXAMPLES--(3) ANOTHER ALTERNATIVE IS TO USE THE MICRO STACK
                                                         AS THE ADDRESS FOR THE LOOP. THE FOLLOWING
                                                         EXAMPLE DOES THE SAME THING AS (1) USING THE
                                                         STACK.
                                 *
 0009   0F03,FFFF,FFFF,FDF4                PUSHLDCNTR FROM '3'; .  *LOAD COUNTER WITH 3 AND PUSH THE NEXT
                                                         MICROADDRESS ONTO THE STACK. THIS ADDRESS WILL
                                                         BE THE LOOPING DESTINATION.*
 000A   040F,FFFF,FFFF,FFFE                CONTINUE; .      *   LOOP   *
 000B   040F,FFFF,FFFF,FFFE                CONTINUE; .      *   BODY   *
 000C   060F,FFFF,FFFF,FFF8                IF NOT CT* THEN LOOP AT FILE:DECREMENT ELSE POP; .  * 'LOOP AT FILE'
                                                         SAYS TO GO TO THE ADDRESS ON THE TOP OF THE
                                                         STACK IF THE COUNTER IS NOT ZERO. 'DECREMENT'
                                                         MUST BE SPECIFIED IF THE LOOP COUNTER NEEDS
                                                         TO BE DECREMENTED. WHEN THE COUNTER IS ZERO,
                                                         THE STACK IS POP'ED IN ORDER TO MAINTAIN IT
                                                         BEFORE GOING TO THE NEXT SEQUENTIAL
                                                         MICROINSTRUCTION.*
```

Figure A23. Looping Examples

- Return from subroutine causes the FILE to be popped, and the microprogram begins execution at the address popped off the FILE.

The FILE, though, is only four levels deep, meaning that subroutines may only be nested four deep before the proper return linkages are destroyed. Figure A24 illustrates use of subroutines.

16-Way Branching. The sequencer also allows branching to one of 2, 4, 8 or 16 consecutive addresses in one instruction. This is done by using the BR MUX and BRANCH CONTROL fields in conjunction with a Jump to PL sequencer instruction. This 16 microword branch table may contain branch instructions, thus allowing a branch to one of 16 routines.

An example of this technique is presented in the Interrupts section below. Note that the branch table for a 16-way branch must begin on a 16-word boundary (i.e. a microaddress ending in 0). Similarly, a 4-way branch table must begin at an address ending in 0, 4, 8, or C; an 8-way branch table must begin at an address ending in 0 or 8; and a 2-way branch table may begin at any even address.

Input/Output (I/O)

The MIME may be microprogrammed for programmed, interrupt driven or Direct Memory Access (DMA) I/O. This section deals strictly with programmed I/O. DMA is discussed in a later section, as is the use of interrupts.

The four least significant bits of the AB are used to select I/O ports. The serial I/O port is addressed as Port 0

```
ADDRESS      MICROCODE           SOURCE PROGRAM

 0011                            PAGE
 0011                            * SUBROUTINE EXAMPLES-- SUBROUTINES ARE DEFINED STARTING WITH A
                                                         LABELED MICROINSTRUCTION. THE 'RETURN'
                                                         MICROSTATEMENT CAUSES A RETURN TO THE
                                                         CALLING ROUTINE. REMEMBER THAT THE SUB-
                                                         ROUTINE STACK IS ONLY 4 LEVELS DEEP AND
                                                         IS CIRCULAR. HERE ARE 2 SUBROUTINES.
                                 *
 0011   040F,FFFF,FFFF,FFFE     SUB1:    CONTINUE;.     *   BODY OF          *
 0012   040F,FFFF,FFFF,FFFE              CONTINUE;.     *   SUBROUTINE 1     *
 0013   0F0F,FFFF,FFFF,FFFA              RETURN;.
 0014   040F,FFFF,FFFF,FFFE     SUB2:    CONTINUE;.     *   BODY OF          *
 0015   040F,FFFF,FFFF,FFFE              CONTINUE;.     *   SUBROUTINE 2     *
 0016   0F0F,FFFF,FFFF,FFFA              RETURN;.
                                 *                      A SUBROUTINE IS INVOKED USING A CALL
                                                        MICROSTATEMENT, AS IN
                                 *
 0017   0F0F,011F,FFFF,FFF1              CALL SUB1;.
                                 *                      IN THE ABOVE CASE, THE LABEL OF THE SUB-
                                                        ROUTINE WAS SPECIFIED IN THE CALL.
                                                        ALTERNATIVELY, THE REGISTER CAN BE FILLED
                                                        AND USED.
                                 *
 0018   040F,014F,FFFF,FFFE              REGISTER = SUB2;.
 0019   1F0F,FFFF,FFFF,FFF5              CALL REGISTER;.
                                 *                      THIS FEATURE MIGHT BE USED TO CONDITIONALLY
                                                        CALL ONE OF TWO SUBROUTINES, AS IN
                                 *
 001A   040F,014F,FFFF,FFFE              REGISTER = SUB2;.
 001B   036F,011F,FFFF,FFF5              IF MZERO THEN CALL SUB1 ELSE CALL REGISTER;. * IN THIS CASE,
                                                        IF MZERO WAS TRUE, THEN 'SUB1' WOULD BE CALLED.
                                                        IF MZERO WAS FALSE, THEN 'SUB2' WOULD BE CALLED.
                                                        SINCE THERE IS ONLY ONE MICROADDRESS FIELD IN
                                                        THE MICROWORD, THIS TYPE OF MANIPULATION IS
                                                        NECESSARY. *
```

Figure A24. Subroutine Examples

for input and Port 1 for output. Ports 2 through F are not implemented but their address decoding is provided. An example of programmed I/O is presented in Figure A25.

DMA

DMA makes use of the BAR to hold the address in memory into or out of which transfer will take place. The WCR is used to hold the two's complement of the number of words remaining to be transferred. At the end of each transfer, the WCR and BAR are incremented. If the BAR overflows, a DMAOVER flag is set. When the WCR reaches 0, the DMATERM flag is set. An example of DMA is given in Figure A26.

Interrupts

The MIME has the capability to service sixteen levels of priority interrupts. Bringing an interrupt line (P0 - P15) low will cause an interrupt request flag to be set if that particular interrupt is of a higher priority than the interrupt currently being serviced (if any). If either of the above conditions are not met, the interrupt is held pending until it is cleared or unmasked, or until it becomes the highest priority interrupt.

An example of a microprogrammed instruction fetch routine with interrupt handler is shown in Figure A27. Note that instructions that direct ICU operation require the Enable Am2914 (code C) command in the same instruction. This example also illustrates the 16-way branching capability of the sequencer.

Parallel Operations

The architecture of MIME allows many operations to be done

```
0019                                    *
0019                                    *PROGRAMMED I/O--A FULL DUPLEX (ECHOING) READ ROUTINE*
0019    0000,FFFF,FFFF,FD1E                 LOAD MAR FROM '0000'!.*PREPARE TO ACCESS PORT 0*
001A    123F,01AF,FFFF,F1F3             WAIT1:  ADDRESS! IF NOT TTYSTATUS THEN GO TO WAIT1!.
                                                *GATE THE MAR ONTO THE AB AND CHECK TO SEE
                                                 IF THE INPUT PORT HAS DATA--IF NOT WAIT FOR IT*
001B    040F,FFFF,FFFF,E1FE                 ADDRESS! IOREAD!.     *GATE MAR ONTO AB AND READ DATA*
001C    040F,FFFF,FFFF,F22E                 LOAD MBR FROM MBR!.*REMEMBER THE MBR'S QUIRK!!!*
001D    0001,FFFF,FFFF,FD1E                 LOAD MAR FROM '0001'!.*PREPARE TO ADDRESS PORT 1*
001E    123F,01EF,FFFF,F1F3             WAIT2:  ADDRESS! IF NOT TTYSTATUS THEN GO TO WAIT2!.
                                                *GATE THE MAR ONTO THE AB AND WAIT FOR THE
                                                 I/O PORT TO BECOME READY*
001F    040F,FFFF,FFFF,D1FE                 ADDRESS! IOWRITE!.   *GATE THE MAR ONTO THE AB AND WRITE DATA*
```

Figure A25. Programmed I/O Example


```
0020
                                        *BLOCK TRANSFER DMA--20H WORDS TO BE TRANSFERRED INTO MEMORY
                                                    BEGINNING AT ADDRESS 0066H
                                        *
0020    0066,FFFF,FFFF,FDAE             DMA!        LOAD BAR FROM '0066'!. *START ADDRESS*
0021    0020,FFFF,FFFF,FD6E                         LOAD DBR FROM '0020'!. *NUMBER OF WORDS*
0022    0407,FFF1,17FF,F6BE                         0 -:1 DBR = 0F! LOAD WCR FROM ALU!. *2'S COMPL OF NO OF WORDS*
0023    0F0F,02AF,FFFF,FFF1             DMALOOP!    CALL READ!. *GET A WORD OF DATA FROM I/O PORT*
0024    040F,FFFF,FFFF,FE2E                         LOAD MBR FROM IOBR!. *MOVE DATA TO MBR*
0025    040F,FFF9,77FF,FA5E                         LOAD MEMORY FROM MBR/DMA!. *GATE BAR ONTO AB,WRITE TO MEMORY*
0026    040F,FFFF,FFFF,2FFE                         INCDMA!.            *INCREMENT THE BAR AND WCR*
0027    020F,029F,FFFF,FFF3                         IF DMAOVER THEN GO TO OVERFLOW!.*CHECK IF ADDRESS HAS OVERFLOWED*
0028    121F,023F,FFFF,FFF3                         IF NOT DMATERM THEN GO TO DMALOOP!.*GO BACK FOR NEXT DATA WORD*
0029                                    * *
0029    040F,FFFF,FFFF,FFFE             OVERFLOW!   CONTINUE!.*DUMMY ERROR HANDLER FOR ADDRESS OVERFLOW*
002A                                    * *
002A    0F0F,FFFF,FFFF,FFFA             READ!       RETURN!. *DUMMY READ ROUTINE*
```

Figure A26. DMA Example

```
002B   127F,030F,FFFF,FFF3        IFTCH:    IF NOT IRQ* THEN GO TO INTHNDL;.
002C   040F,FFFF,FFFF,F01E                  LOAD MAR FROM PC;. *GET ADDRESS OF NEXT INSTRUCTION*
002D   040F,FFF9,77FF,014E                  INCPC; LOAD MBR FROM MEMORY;.
002E   040F,FFFF,FFFF,F23E                  LOAD IR FROM MBR;.
002F   040F,FFFF,FFFF,FFF2                  GO TO MAP;. *OUTPUT OF MAPPING PROM IS MICROADDRESS
                                                        OF ROUTINE TO DECODE/EXECUTE INSTRUCTION
                                  *
0030                                        * INTERRUPTS - SAVE OLD INTERRUPT VECTOR, PUT IN NEW ONE *
0030                                        * SELECT PROPER INTERRUPT HANDLER *
0030   5F0F,040F,FF5F,CFFF        INTHNDL:  ON IV0,IV1,IV2,IV3 GO TO INTBR;.
0040   040F,060F,FFFF,FFFF        INTBR:16  GO TO INTF;. *BRANCH TABLE*
0041   040F,05FF,FFFF,FFFF                  GO TO INTE;.
0042   040F,05EF,FFFF,FFFF                  GO TO INTD;.
0043   040F,05DF,FFFF,FFFF                  GO TO INTC;.
0044   040F,05CF,FFFF,FFFF                  GO TO INTB;.
0045   040F,05BF,FFFF,FFFF                  GO TO INTA;.
0046   040F,05AF,FFFF,FFFF                  GO TO INT9;.
0047   040F,059F,FFFF,FFFF                  GO TO INT8;.
0048   040F,058F,FFFF,FFFF                  GO TO INT7;.
0049   040F,057F,FFFF,FFFF                  GO TO INT6;.
004A   040F,056F,FFFF,FFFF                  GO TO INT5;.
004B   040F,055F,FFFF,FFFF                  GO TO INT4;.
004C   040F,054F,FFFF,FFFF                  GO TO INT3;.
004D   040F,053F,FFFF,FFFF                  GO TO INT2;.
004E   040F,052F,FFFF,FFFF                  GO TO INT1;.
                                  * *
```

Figure A27. Interrupt Routine Example

```
004F                                        # INTERRUPT HANDLERS #
004F   040F,FFFF,FF4F,CFFE        INT0:     CLEARVEC;. #CLEAR THE VECTOR FOR THIS INTERRUPT#
0050   0F0F,061F,FFFF,FFF1                  CALL PUTOLD;. #SAVE THE OLD PROGRAM STATUS#
0051   040F,062F,FFFF,FFFF                  GO TO GETNEW;. #GET THE NEW (INT HANDLER) STATUS#
0052                                        #THE REST OF THE INTERRUPT HANDLERS ARE SIMILAR TO ABOVE#
0052   040F,FFFF,FFFF,FFFE        INT1:     CONTINUE;.
0053   040F,FFFF,FFFF,FFFE        INT2:     CONTINUE;.
0054   040F,FFFF,FFFF,FFFE        INT3:     CONTINUE;.
0055   040F,FFFF,FFFF,FFFE        INT4:     CONTINUE;.
0056   040F,FFFF,FFFF,FFFE        INT5:     CONTINUE;.
0057   040F,FFFF,FFFF,FFFE        INT6:     CONTINUE;.
0058   040F,FFFF,FFFF,FFFE        INT7:     CONTINUE;.
0059   040F,FFFF,FFFF,FFFE        INT8:     CONTINUE;.
005A   040F,FFFF,FFFF,FFFE        INT9:     CONTINUE;.
005B   040F,FFFF,FFFF,FFFE        INTA:     CONTINUE;.
005C   040F,FFFF,FFFF,FFFE        INTB:     CONTINUE;.
005D   040F,FFFF,FFFF,FFFE        INTC:     CONTINUE;.
005E   040F,FFFF,FFFF,FFFE        INTD:     CONTINUE;.
005F   040F,FFFF,FFFF,FFFE        INTE:     CONTINUE;.
0060   040F,FFFF,FFFF,FFFE        INTF:     CONTINUE;.
                                  # #
0061   040F,FFFF,FFFF,FFFE        PUTOLD:   CONTINUE;.#DUMMY PROGRAM SEGMENT-
                                                  PUTOLD MUST SAVE THE OLD PROGRAM STATUS
                                  #
0062   040F,FFFF,FFFF,FFFE        GETNEW:   CONTINUE;. #DUMMY PROGRAM SEGMENT-
                                                GETNEW MUST LOAD INTERRUPT HANDLER
                                                  PROGRAM STATUS
                                  #
0063   040F,02BF,FFFF,FFFF                  GO TO IFTCH;. #FETCH THE FIRST INSTRUCTION OF
                                                THE INTERRUPT HANDLER
                                                 #
```

Figure A27 (Continued)

in parallel with other operations. In general, any operation may be done in parallel with any other operation(s) except when two operations require use of the same microword field (e.g. incrementing the PC and loading the B latch simultaneously is not allowed since both operations must use the COMMAND field).

IV. In Case of Difficulty

This chapter provides general guidance to the user to help discover the cause if MIME will not operate correctly. The procedures given will correct the problems most often encountered by the authors during their work on MIME. The following steps should be followed:

1. Turn ON/OFF switch to OFF.
2. Check switch positions on circuit cards.
   a. CS1 (card C) - RAM/ROM switch to ROM (away from front panel)
   b. CCU1 (card D) - Dip switches
      SW1 - OFF
      SW2 - ON
      SW3 - ON
      SW4 - OFF
   c. I/O (card H) - Dip switches
      SW1 - OFF
      SW2 - 10 - Select desired baud rate
3. Check all circuit cards firmly seated in backplane connectors.
4. Check all ribbon cables properly installed between FP and circuit cards.
5. Check all ribbon cables properly installed between circuit cards.
   a. Card C to card D

b. Card E to card F

c. Card G to card H

6. Check cable to terminal. Secure at both ends.

7. Check terminal switches.

   a. Baud rate - to match baud rate set in 2.c.

   b. Full duplex

   c. 8 bits data

   d. No parity

   e. 2 stop bits

8. Set front panel switches

   a. All momentary switches (e.g. HALT/RUN) in center position.

   b. All positive action switches (e.g. BPI) in down (OFF) position.

   c. ON/OFF to ON

9. Upon power up the following should occur:

   a. The power relay should engage with an audible "clunk". If not, check:

      1) Power cord connected to 110 VAC

      2) Power line fuse not blown

      3) ON/OFF switch good

      4) Power relay

   b. The cooling fans should start. If not, check power relay.

   c. The following front panel indicators should light:

      1) HALT

      2) PC

3) A

4) MACRO DISPLAY - all 0's

5) MICRO DISPLAY - 3 0's

If not check +5V power supply and fuses.

10. Turn terminal on and allow to warm up (if necessary).

11. Depress HALT/RUN to RUN and release

    a. The HALT light should extinguish.

    b. The RUN light should illuminate.

12. Assuming that the MIME/MM ROM is installed, the following should be printed on the terminal:

```
MIME/MM
COM?
```

13. If step 12 met with success, try the DR and LR functions of MIME/MM (see Appendix D).

14. If steps 12 and 13 worked, the MEM, ALU, CS1, CCU1, CCU2 and I/O boards are working essentially correctly.

15. If step 12 worked but not step 13, the MEM and ALU boards would be suspect. Go to step 17.

16. If neither 12 nor 13 worked, the CS1, CCU1, CCU2, and I/O boards would be suspect. Continue with step 17.

17. To isolate the I/O board, try the Memory Diagnostic.

    a. Reset MIME.

    b. Select MACRO DISPLAY of MAR.

    c. Select MICRO on MICRO/MACRO switch (DATA ENTRY).

    d. Load the Diagnostic start address (0362).

    e. Momentarily press RUN.

    f. The HALT light should momentarily extinguish and

the Run light should illuminate momentarily.

g. When the HALT light reilluminates, the MACRO DISPLAY shows the first bad address in memory.

18. If 17 was successful while 12 was not, the problem is in the I/O board.

19. If step 17 failed, the most basic of tests is in order:

a. Reset MIME.

b. Set RAM/ROM switch on card C to RAM (towards front panel).

c. Select CS on MICRO DISPLAY.

d. Select MICRO on MICRO/MACRO switch.

e. Enter the following:

FFFF FFFF FFFF FFFE

LOAD NEXT

FFFF FFFF FFFF FFFE

LOAD NEXT

FFFF FFFF FFFF FFF0

LOAD

f. Reset MIME.

g. Set MSTEP CLOCK RATE to its minimum setting.

h. Press the following control switches:

1) AUTO on AUTO/MSTEP switch

2) RUN

i. The MICRO display should show the following sequence:

000

001

002

000

001

etc.

20. The test above used only FP, CS1 and CCU1, hence if the test failed, these boards are the suspects.

21. For details of individual module operation, refer to Chapter I of this appendix. Schematic diagrams of all modules are contained in Volume III.

List of References

1. Purvis, Richard E. and Ronald D. Yoho. MIME: Microprogrammable Minicomputer Emulator. Unpublished thesis. Wright-Patterson Air Force Base: Air Force Institute of Technology, March, 1978.

2. Mick, John R. and Jim Brick. Microprogramming Handbook and Am2910 Emulation. (Second Edition) Sunnyvale: Advanced Micro Devices, 1977.

3. Am2900 Bipolar Microprocessor Family. Product Brochure. Sunnyvale: Advanced Micro Devices, 1976.

4. Am2914 Priority Interrupt Encoder. Product Brochure. Sunnyvale: Advanced Micro Devices, 1976.

Appendix B

MIL-STD-1750 Emulation Reference Manual

Preface

This manual describes the additional hardware and the emulation microcode that were developed to emulate the MIL-STD-1750 instruction set. It is assumed in this manual that the user is familiar with the structure of MIME, including the names of the registers and the basic MIME operations. This information is available in Appendix A, MIME User's Manual.

Contents

List of Tables

List of Figures

List of Terms

A latch	A address latch
ALU	Arithmetic Logic Unit
Am2900	Advanced Micro Devices 2900 series circuits
AUX	Refers to auxiliary module
AUXn	Auxiliary Register number n
B latch	B address latch
BR	Base Register
DB	Data Bus
DBR	Data Buffer Register
DO	Derived Operand
FR	Fault Register
IR	Instruction Register
MIL-STD-1750	Airborne Computer Instruction Set Architecture
MIME	Microprogrammable Minicomputer Emulator
MR	Mask Register on Am2914
MUX	Multiplexer
O.C.	Operation Code (opcode)
OCX	Opcode extension
opcode	Operation Code
Pnn	Interrupt priority number nn (MIME)
PC	Program Counter
Q	Am2901 Q register
RA, RB	Operands in MIL-STD-1750
Rx	Index register in MIL-STD-1750

SW	Status Word
UDACn	User Definable Auxiliary Command number n
UDAFn	User Definable Auxiliary Function number n
UDTCn	User Definable Test Code number n
(register)	Contents of register

I. Introduction

This manual describes the hardware and software configuration that was created to emulate the MIL-STD-1750 instruction set. The hardware provides features not found in MIME that were required to conform to the definition of the instruction set. The software (emulation microcode) is an implementation of the register transfer language given in MIL-STD-1750. The realization of some example instructions is explained in detail.

II. Use of MIME Features in Emulation

This chapter summarizes the machine architecture features required by MIL-STD-1750, and indicates the MIME capabilities used to meet these requirements. Table BI provides a list of the features as they are realized in the MIME/MIL-STD-1750 emulation, classing them either as part of the original MIME, the enhanced MIME, the emulation-specific hardware or the emulation software. Reference 1 discusses the rationale behind each of these realizations.

AD-A080 420 AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OH SCHOO--ETC F/G 9/2
MIME: MICROPROGRAMMABLE MINICOMPUTER EMULATOR. PHASE II. VOLUME--ETC(U)
DEC 79 T R HOYT, D A MYERS
UNCLASSIFIED AFIT/GCS/EE/79-11-VOL-2 NL

2 OF 3
AD
A080420

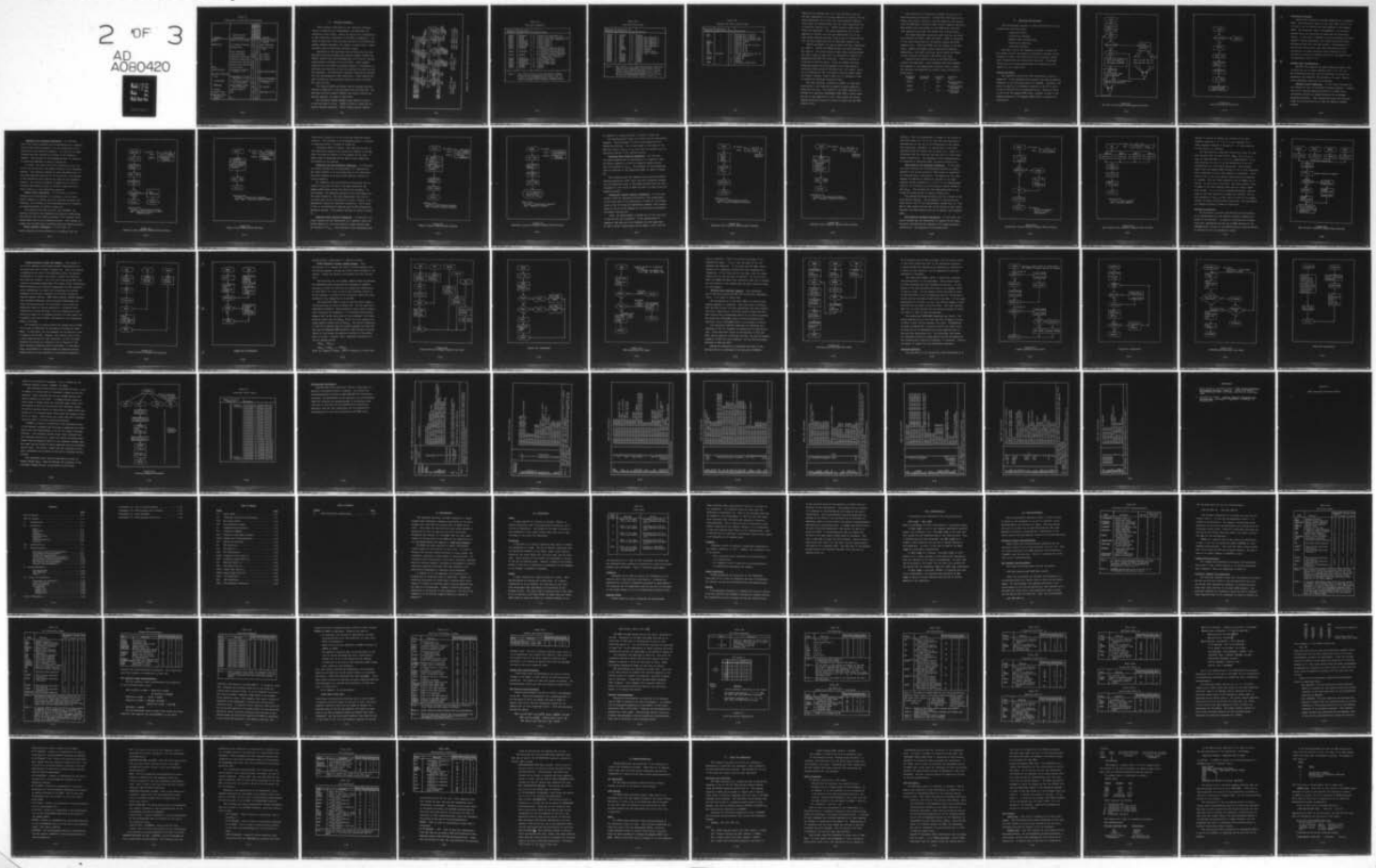


Table BI

Realization of MIL-STD-1750 Features

Feature	MIL-STD-1750 Requirement	Original MIME	Enhanced MIME	Emulation H/W	Emulation S/W	Remarks
Memory	64K x 16 bit words	X				1K words implemented
Registers	16 General Purpose	X				ALU registers
	Status Word			X		Derived from CCR
	Fault Register			X		
	Interrupt Mask	X	X			On Am2914
	Pending Interrrupts	X	X			On Am2914
	Auxiliary Registers			X		Six added
	Instruction Counter	X				
	Instruction Register	X				
Data	16 bit words	X				
Instructions	Arithmetic,logical control, bit				X	Microprogrammed
Special Features						
I/O	Programmed I/O from registers to port	X			X	Microprogrammed
Interrupts	16 levels of priority interrupts	X	X			Expanded from 8 to 16
Masking	Hold masked interrupts pending	X	X			On Am2914
Timers	Timers A and B			X		
Control Store	N/A	X	X			Expanded to 4K
Auxiliary Test Condition Flags	N/A			X		

III. Emulation Hardware

Those features identified in the "emulation hardware" column of table BI were implemented on the MIL-STD-1750 auxiliary circuit board. Figure B1 depicts the configuration of this board and a schematic is included in Appendix A. As shown in figure B1, this aux board consists of six additional general purpose registers, two timers, a status word, a fault register, and two additional condition flags.

Operations on the aux board are controlled by the user-definable auxiliary command and function signals (UDACn* and UDAFn*) derived from the corresponding bits in the PL through demultiplexers external to the aux board. These signals interact with the AUXRD* and AUXLD* signals, if necessary, to determine the direction of data flow between the data bus and the registers. The additional condition flags may be queried from the microprogram as test conditions. These signals and their relation to the microword fields are shown in tables BII, BIII and BIV.

The signals AUXRD* and AUXLD* control reading from and writing to registers on the aux board from the data bus. The auxiliary function signals (UDAFn*) are used to select the desired register, as shown in table BIII.

The auxiliary command signals cause specific actions on the aux board to occur. UDACØ* is used to clear the six general purpose registers. UDAC1* causes general purpose

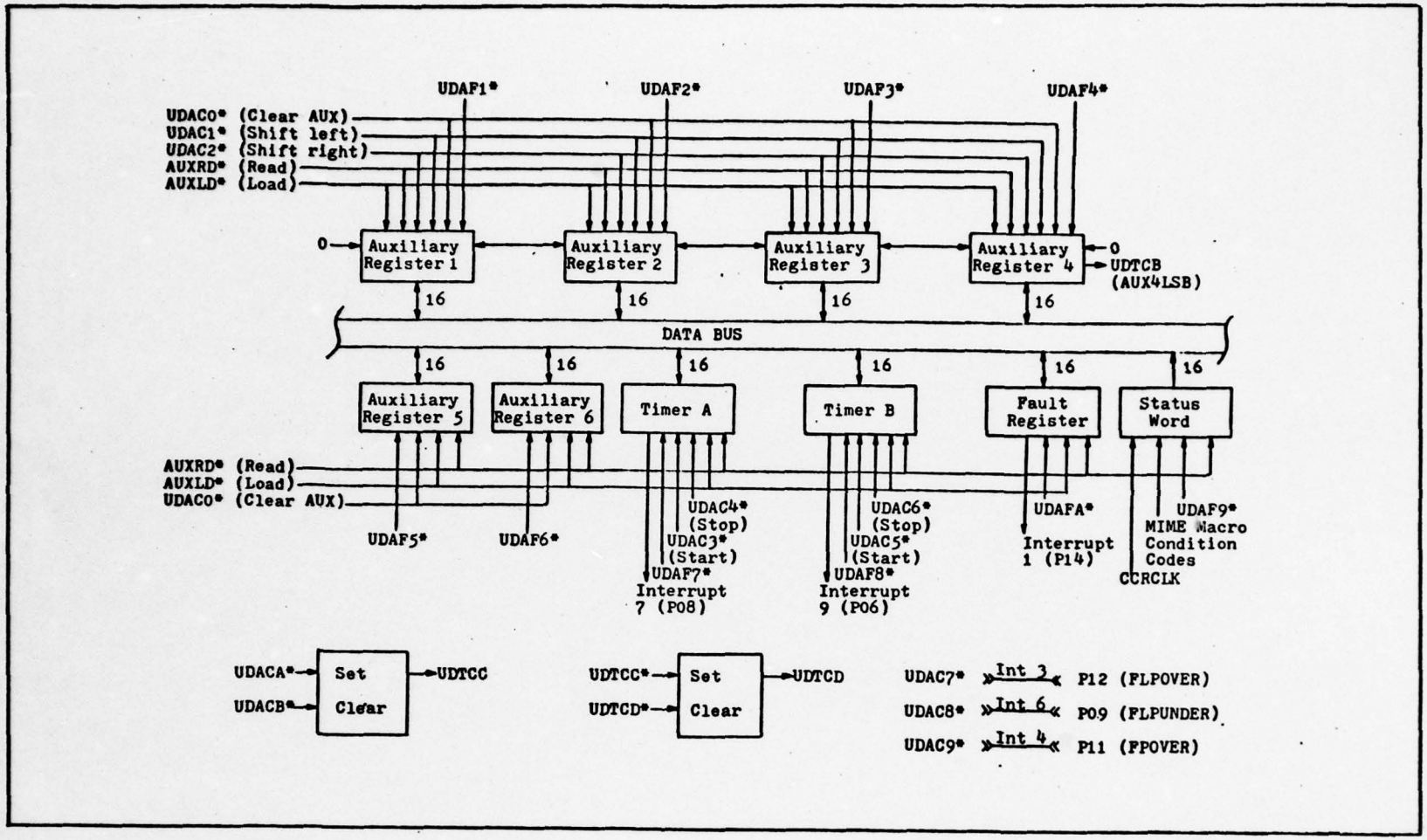


Figure B1. MIL-STD-1750 Auxiliary Board Block Diagram

Table BII

Auxiliary Commands[1]

Translator Mnemonic	Translator Redefinition	Code	Function
UDACØ	CLEARAUX	Ø	Clear aux registers 1-6
UDAC1	SLAUX	1	Shift left aux registers 1-4
UDAC2	SRAUX	2	Shift right aux registers 1-4
UDAC3	STARTTIMA	3	Start timer A
UDAC4	STOPTIMA	4	Stop timer A
UDAC5	STARTTIMB	5	Start timer B
UDAC6	STOPTIMB	6	Stop timer B
UDAC7	FLPOVER	7	Generate interrupt #3 for floating point overflow
UDAC8	FLPUNDER	8	Generate interrupt #6 for floating point underflow
UDAC9	FPOVER	9	Generate interrupt #4 for fixed point overflow
UDACA	SETUDTCC	A	Set UDTCC flag
UDACB	CLEARUDTCC	B	Clear UDTCC flag
UDACC	SETUDTCD	C	Set UDTTCD flag
UDACD	CLEARUDTCD	D	Clear UDTCD flag
UDACE	UNUSED	E	Spare
UDACF	UNUSED	F	Spare

Note 1 - The "code" is set in the auxiliary command field of the microword, bits 23-20. An "F" is set in the command field, bits 15-12, in order to enable the auxiliary command field.

Table BIII

Auxiliary Functions[1]

Translator Mnemomic	Translator Redefinition	Code	Function
UDAFØ	UNUSED	Ø	Spare
UDAF1	AUX1	1	Select aux register 1
UDAF2	AUX2	2	Select aux register 2
UDAF3	AUX3	3	Select aux register 3
UDAF4	AUX4	4	Select aux register 4
UDAF5	AUX5	5	Select aux register 5
UDAF6	AUX6	6	Select aux register 6
UDAF7	TIMERA	7	Select timer A
UDAF8	TIMERB	8	Select timer B
UDAF9	SW	9	Select status word
UDAFA	FR	A	Select fault register
UDAFB	UNUSED	B	Spare
UDAFC	UNUSED	C	Spare
UDAFD	UNUSED	D	Spare
UDAFE	UNUSED	E	Spare
UDAFF	UNUSED	F	Spare

Note 1 - The "code" is set into the auxiliary function field of the microword, bits 19-16. For a read of the selected register, a "5" is set in the bus source field, bits 11-8. For a write to the selected register, a "D" is set in the bus destination field, bits 7-4.

Table BIV

Updated CCU Test Condition Mux

Translator Mnemonic	Translator Redefinition	Code	Condition
BP1	-	Ø	Breakpoint 1 switch
BP2	-	1	Breakpoint 2 switch
-	-	2	Select I/O condition mux
-	-	3	Select ALU condition mux
MARØ	-	4	Bit Ø of MAR
MEMORY	-	5	Memory FF
CT*	-	6	Loop counter=Ø if false
IR4	-	7	Bit 4 of IR
IR5	-	8	Bit 5 of IR
IR7	-	9	Bit 7 of IR
IR15	-	A	Bit 15 of IR
UDTCB	AUX4LSB	B	Least significant bit of aux register 4
UDTCC	-	C	UDTCC FF
UDTCD	-	D	UDTCD FF
-	-	E	Spare
TRUE	-	F	Constant high level

registers one through four to be left shifted by one bit. The most significant bit of each register is shifted into the least significant bit of the next lower numbered register, with a zero bit being shifted into the least significant bit of auxiliary register four. UDAC2* causes a right shift of these four registers. The least significant bit of each register is shifted into the most significant bit of the next higher numbered register, with a zero bit being shifted into the most significant bit of auxiliary register one.

UDAC3* and UDAC4* are used to start and stop, respectively, timer A. UDAC5* and UDAC6* perform the same functions for timer B. The carry output of each timer (generated when the timer counts up past zero) is connected to produce the appropriate MIL-STD-1750 interrupt. Timer A produces an interrupt of priority seven. Since the Am2914 interrupt numbering assigns the high numbers to the high priority interrupts (the reverse of MIL-STD-1750), the timer A carry out is connected to P08, the Am2914 interrupt number eight. For similar reasons, timer B carry out is connected to P06 to produce an interrupt of priority nine.

The fault register, selected by UDAFA*, is read from and written to the same way as general purpose registers, using the data bus. The outputs of the fault register are decoded by a specially programmed 74387 PROM to detect when any bit in the register is a one. This causes the second highest priority interrupt to occur by connecting the PROM output to P14.

The status word is selected by UDAF9* and can only be read from using the data bus. The MIL-STD-1750 flags in the status word (carry, positive, zero and negative) are derived from the MIME macro condition codes (carry, negative and zero). When CCRCLK (the load signal for the MIME macro condition code register) goes high, the status word is also loaded.

The three additional condition codes from the aux board are available as UDTCB, UDTCC and UDTCD. UDTCB is connected to the least significant bit of general purpose auxiliary register four. UDTCC and UDTCD are the outputs of two flip flops. UDACA* and UDACB* control setting and clearing, respectively, the UDTCC flip flop, while UDACC* and UDACD* control the same functions on the UDTCD flip flop.

Underflow and overflow errors in MIL-STD-1750 are signaled by interrupts. User definable auxiliary commands are jumpered to the appropriate interrupt lines to enable the emulation microprogram to generate those interrupts as shown below:

Auxiliary Command	Interrupt Priority	Interrupt Signal	Interrupt Cause
UDAC7*	3	P12	Floating point overflow
UDAC8*	6	P09	Floating point underflow
UDAC9*	4	P11	Fixed point overflow

IV. Emulation Microprogram

The microprogram designed to emulate MIL-STD-1750 can be divided into five main parts:

-instruction fetch

-instruction decoding

-address mode interpretation

-instruction execution

-interrupt handling

The basic flow of the emulation is shown in figure B2. The normal flow for emulating an instruction consists of performing, in order, the instruction fetch, decoding, address mode interpretation and instruction execution. Interrupts which occur on a non-scheduled basis are handled as an exception to this normal flow.

Instruction Fetch.

The emulation instruction fetch processing is shown in figure B3. If an interrupt has occurred, the fetch processing transfers to the interrupt handler at INTHNDL. Otherwise, the value in memory at the address contained in the PC is moved to the IR and the PC is incremented by one. Using the value in the IR as the mapping PROM input, transfer is made to the address produced at the mapping PROM output to execute the instruction.

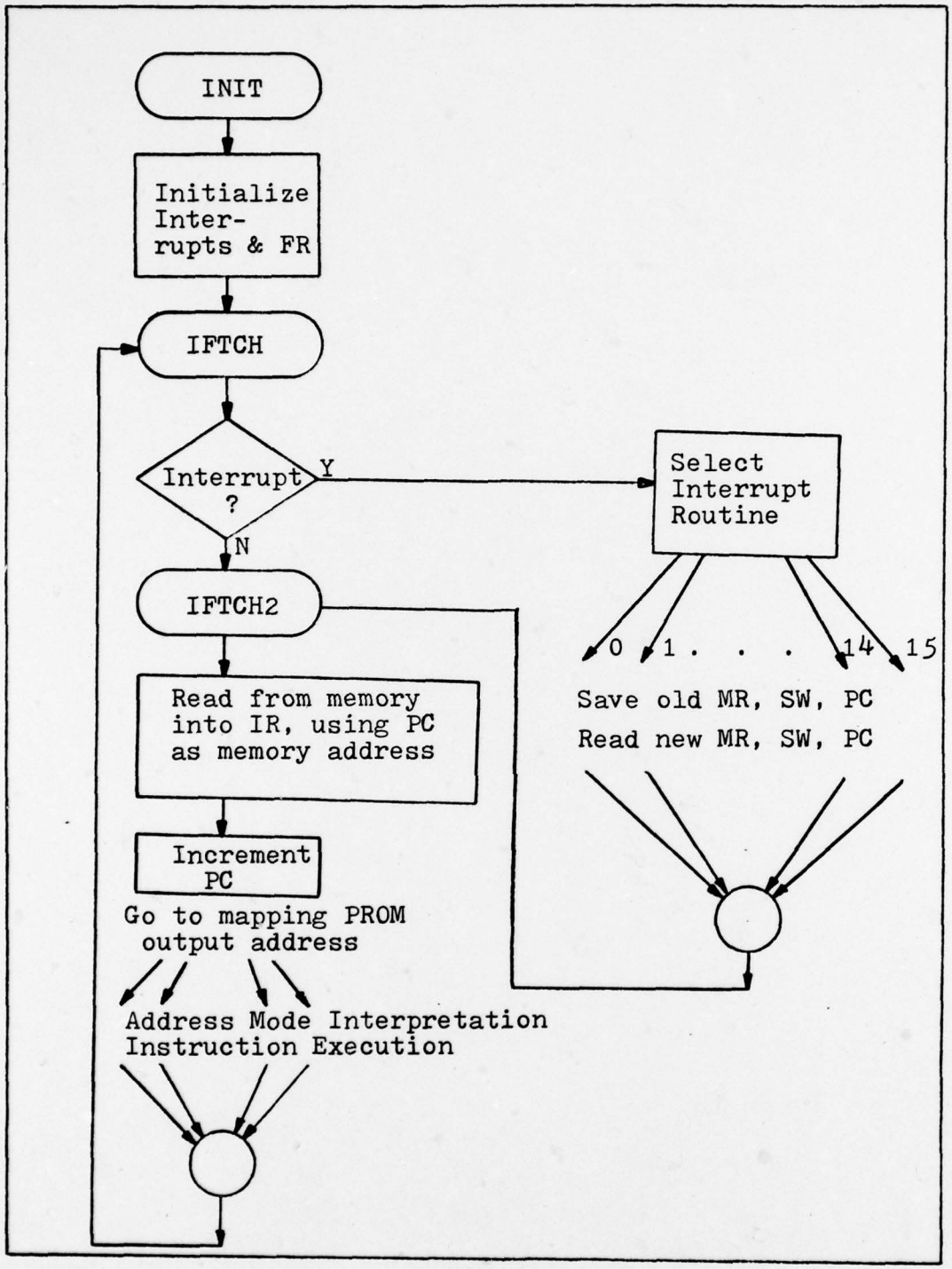


Figure B2.
MIL STD 1750 Emulation Microprogram Flowchart

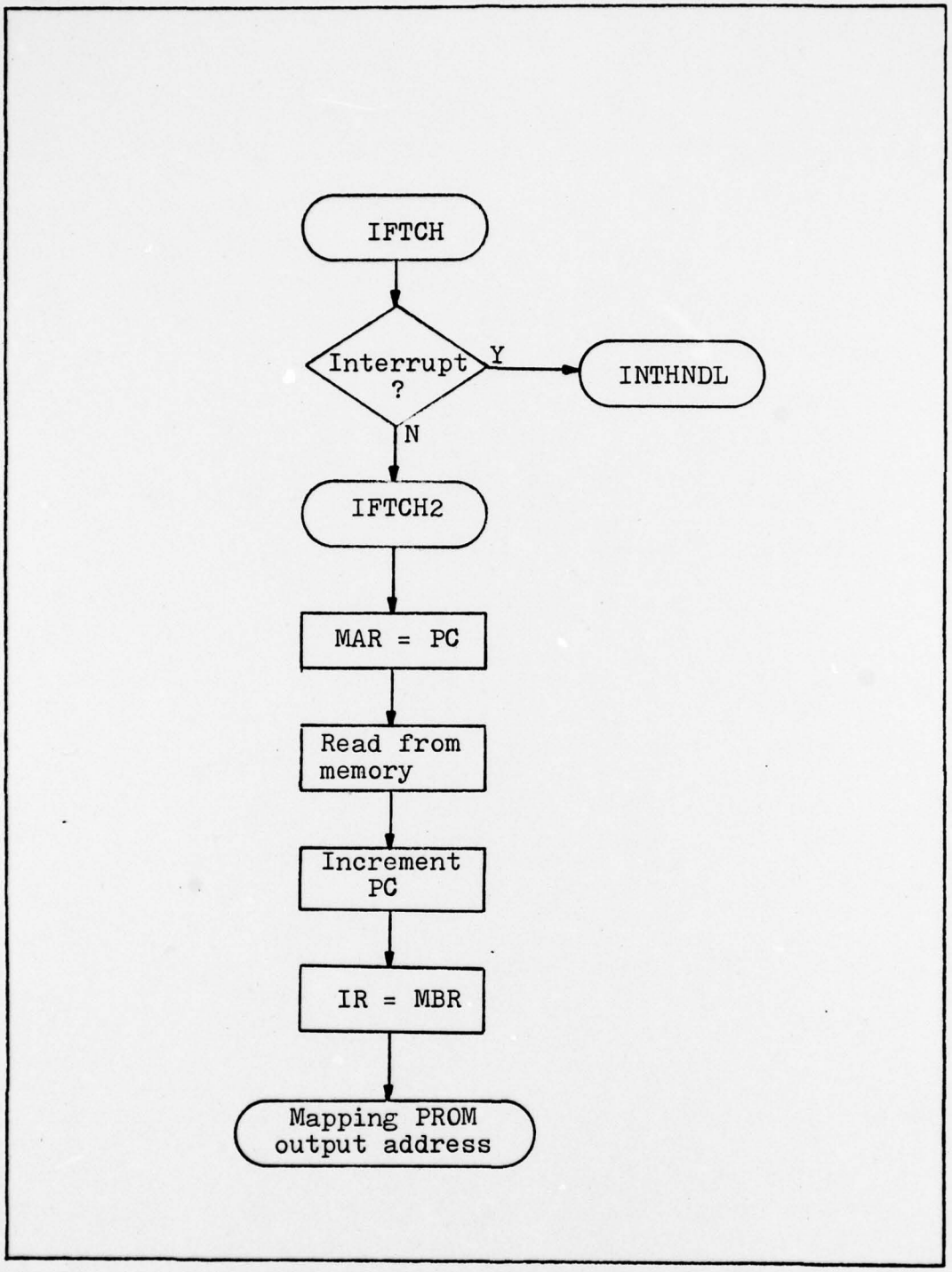


Figure B3.
Instruction Fetch Flow Chart

Instruction Decoding

Instruction decoding is handled primarily by the mapping PROM. The instruction opcode in the high order byte of the IR (Ref 2: 22) is used as the input address of the mapping PROM. The resulting output is programmed to be the micro-address where the microcode for executing the instruction begins. In the case of immediate long non-indexible and base register indexed addressing, all instructions using these addressing modes have the same opcode. Differentiation of the various instructions with these addressing modes is governed by an opcode extension found elsewhere in the instruction word and is handled as part of the address mode interpretation (Ref 2: 23).

Address Mode Interpretation

MIL-STD-1750 supports ten basic addressing modes, some of which may use an index register (Ref 2: 16 - 17, 21). Each addressing mode has its own subroutine to decode the addressing and determine the operands to be used. Each of these addressing modes will be discussed.

Register Direct Addressing. In this mode, operands for the instruction are in the general purpose registers. Loading the A and B register address latches is a single micro-instruction and was not deemed suitable for an address decoding subroutine. Each instruction using this mode performs the microinstruction to load the register address latches.

Immediate Long Indexible Addressing. In this mode, one of the source operands and the destination is a register, and the other source operand is in the second word of the instruction (the immediate operand.) The value in an index register may also optionally be added to the immediate operand. The decoding of this addressing mode is contained in subroutine $IM/IMX, as shown in figure B4.

On entry, the first word of the instruction is in the IR and the PC points to the memory location of the immediate operand. The immediate operand is read from memory and, if the index register field of the IR is non-zero, the contents of the specified index register are added to the immediate operand. The resulting operand is passed back to the instruction processing routine in the DBR, along with the B latch being set to the destination register.

Memory Direct Addressing. In this mode, one source operand and the destination is a register, and the other source operand is a memory location, possibly accessed with indexing. The decoding of this addressing mode is contained in subroutine $D/DX, as shown in figure B5.

This address decoding initially calls $IM/IMX, the decoding subroutine for immediate long indexible addressing. This returns with the absolute address of the operand, after having taken care of any indexing involved. This address is loaded into the MAR before returning from this address decoder.

Memory Indirect Addressing. In this mode, one source operand and the destination is a register, and the

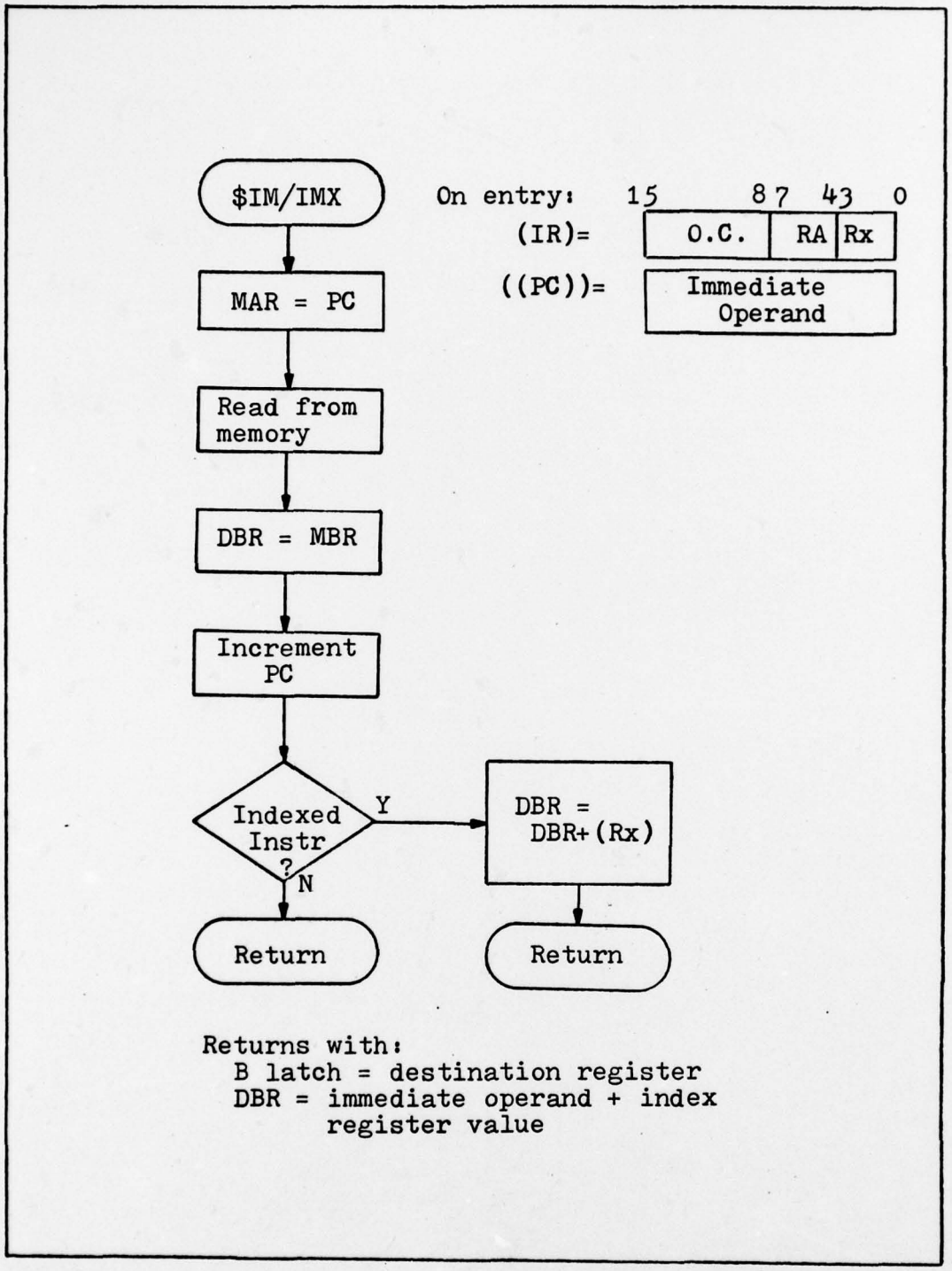


Figure B4.
Immediate Long Indexible Address Mode Decoding

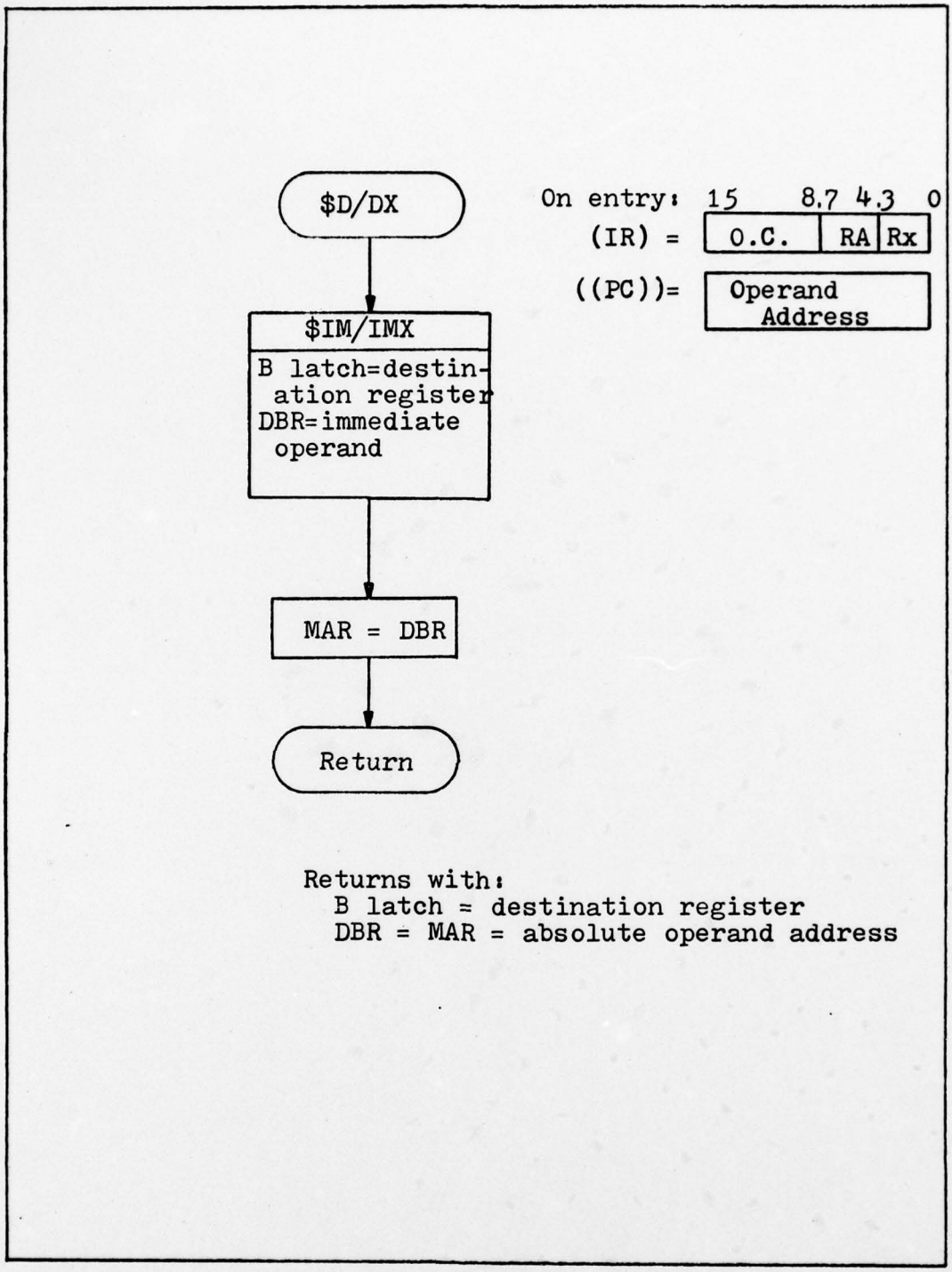


Figure B5.
Memory Direct Address Mode Decoding

other source operand is in an indirectly addressed memory location. The decoding of this addressing mode is contained in subroutine $I/IX, as shown in Figure B6.

Initially, $D/DX is called. This sets the B latch to the destination register and returns the address in the MAR where the absolute address of the operand can be found. A memory read is performed and the MAR is then loaded with the address of the operand.

Immediate Long Non-Indexible Addressing. In this mode, one source operand and the destination is a register and the other operand is in the second word of the instruction. The field normally used by an index register contains an opcode extension.

All immediate long non-indexible instructions have an opcode of $4A_{16}$ and are sent to the same address by the mapping PROM output during the instruction decoding. At this address, the immediate operand is read from memory and the opcode extension (the least significant four bits of the first word of the instruction) is used to branch to the appropriate instruction execution subroutine. The B latch is set to the destination register and the DBR contains the immediate operand. This address decoding is shown in Figure B7.

Immediate Short Positive Addressing. In this mode, one source operand and the destination is a register, while the second operand is a four-bit positive number derived from the contents of IR_{0-3}. The decoding of this addressing mode

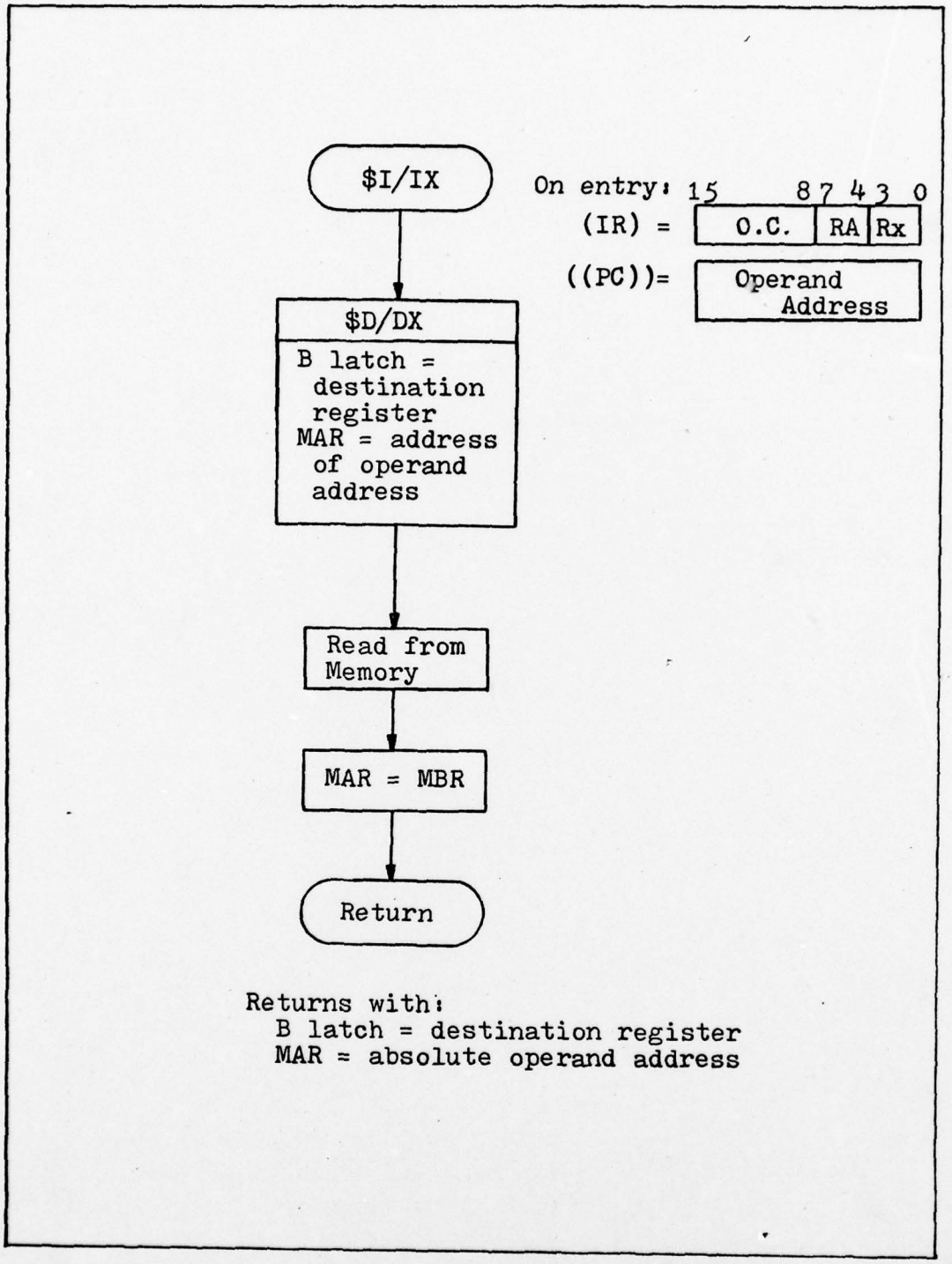


Figure B6.
Memory Indirect Address Mode Decoding

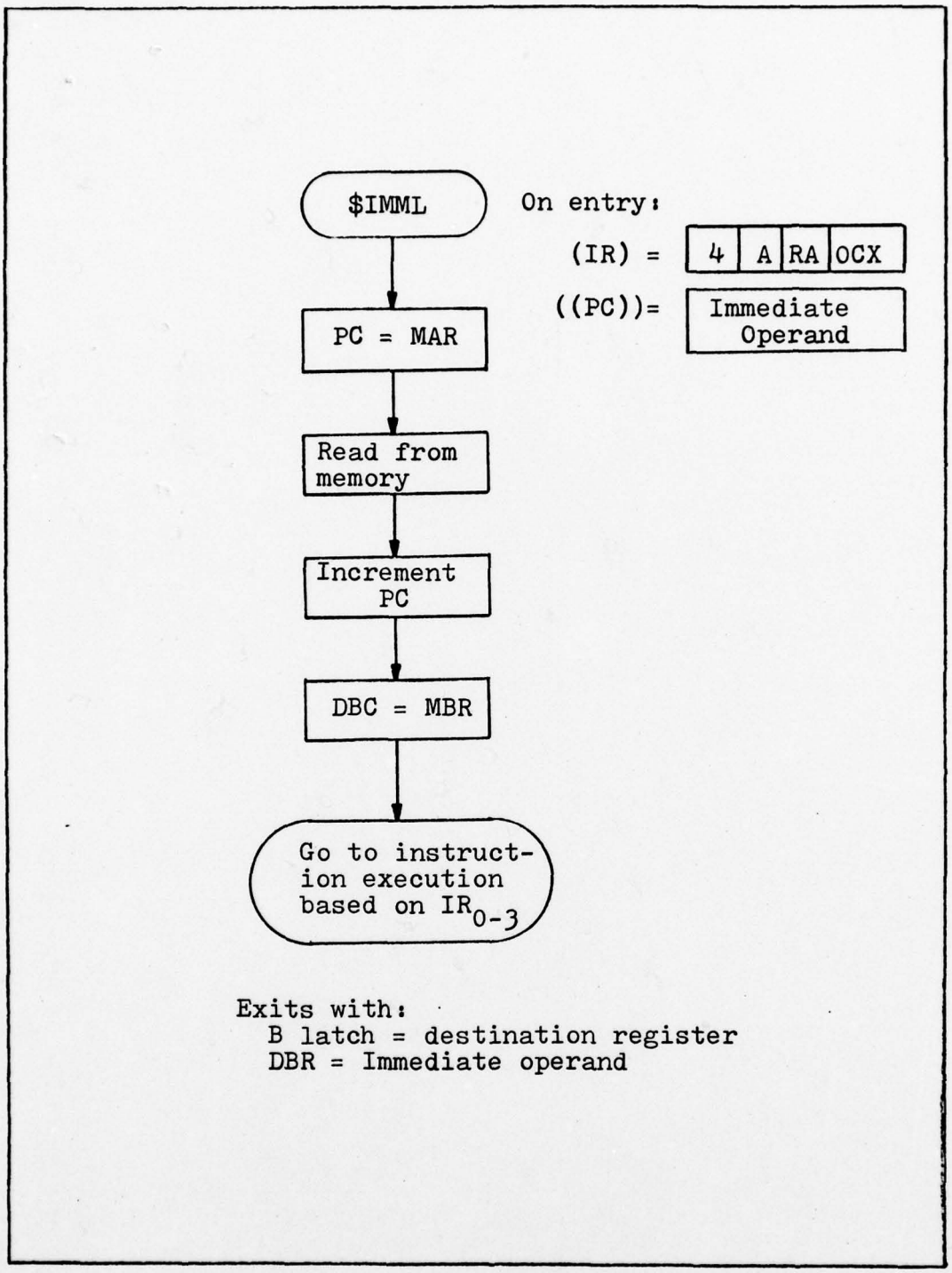


Figure B7.
Immediate Long Non-Indexible Address Mode Decoding

is contained in subroutine $ISP, as shown in Figure B8.

The decoding first loads the B latch with the destination register. Bits Ø through 3 of the IR are masked off and loaded into the DBR. One is then added to the value in the DBR (the value in IR_{0-3} is one less than the operand) and the subroutine returns to its calling routine.

Immediate Short Negative Addressing. In this mode, one source operand and the destination is a register, while the second operand is a four-bit negative number derived from the contents of IR_{0-3}. The decoding of this addressing mode is contained in the subroutine $ISN, as shown in Figure B9.

This decoding uses the immediate short positive address decoding subroutine, $ISP, since the only difference between the two addressing modes is that $ISN requires that the two's complement of the result of $ISP be taken in order to get the negative operand.

Instruction Counter Relative Addressing. In this mode, which is used for branching instructions, the displacement in the lower byte of the instruction is added to the current value of the PC to get the destination address. (The current value of the PC is the address of the instruction following the branch instruction).

First, the displacement is masked out of the lower byte of the IR into the Q register. If the displacement is negative, the sign has to be extended into the upper byte so that a 16-bit displacement can be added to the 16-bit PC

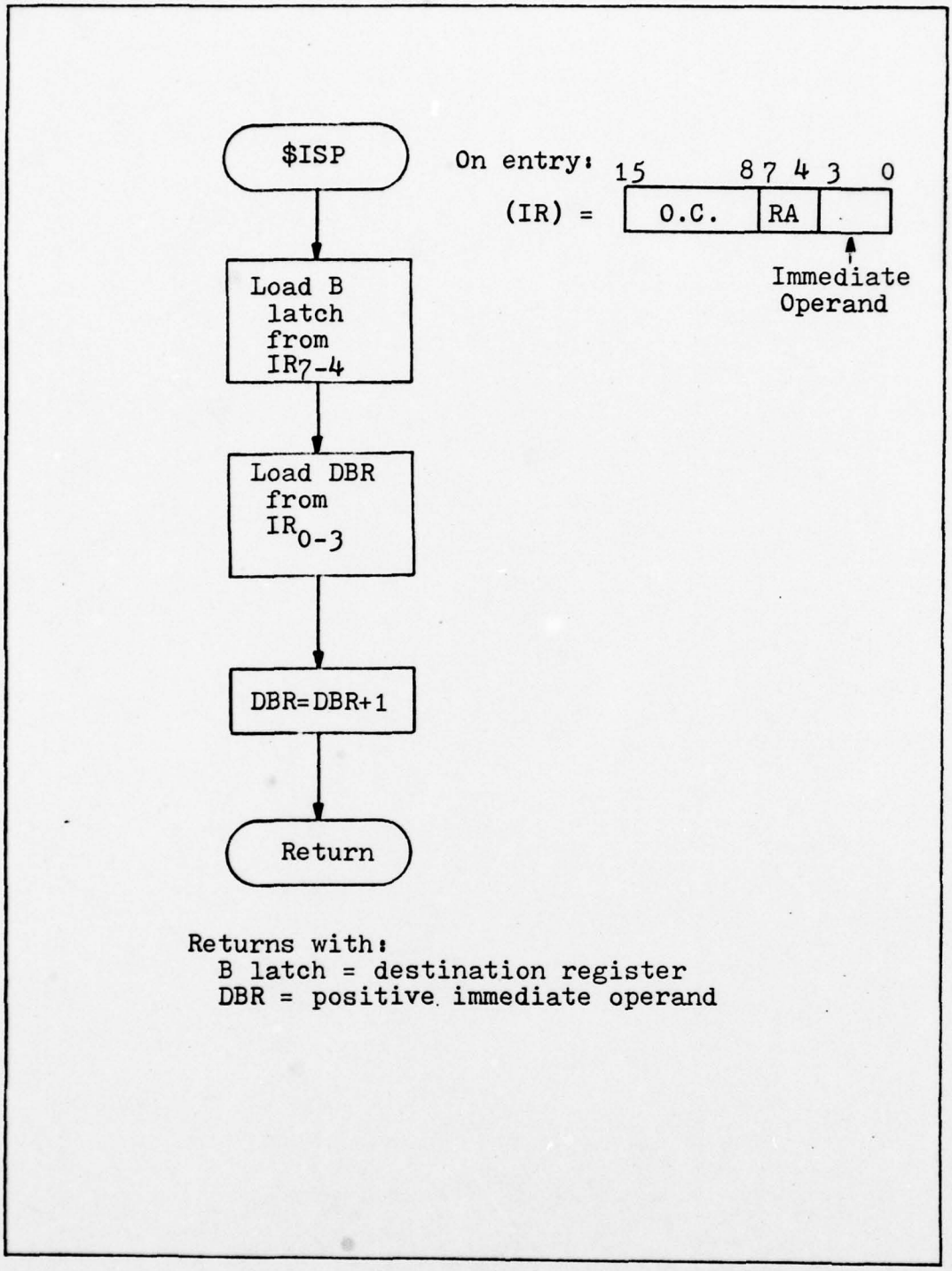


Figure B8.
Immediate Short Positive Address Mode Decoding

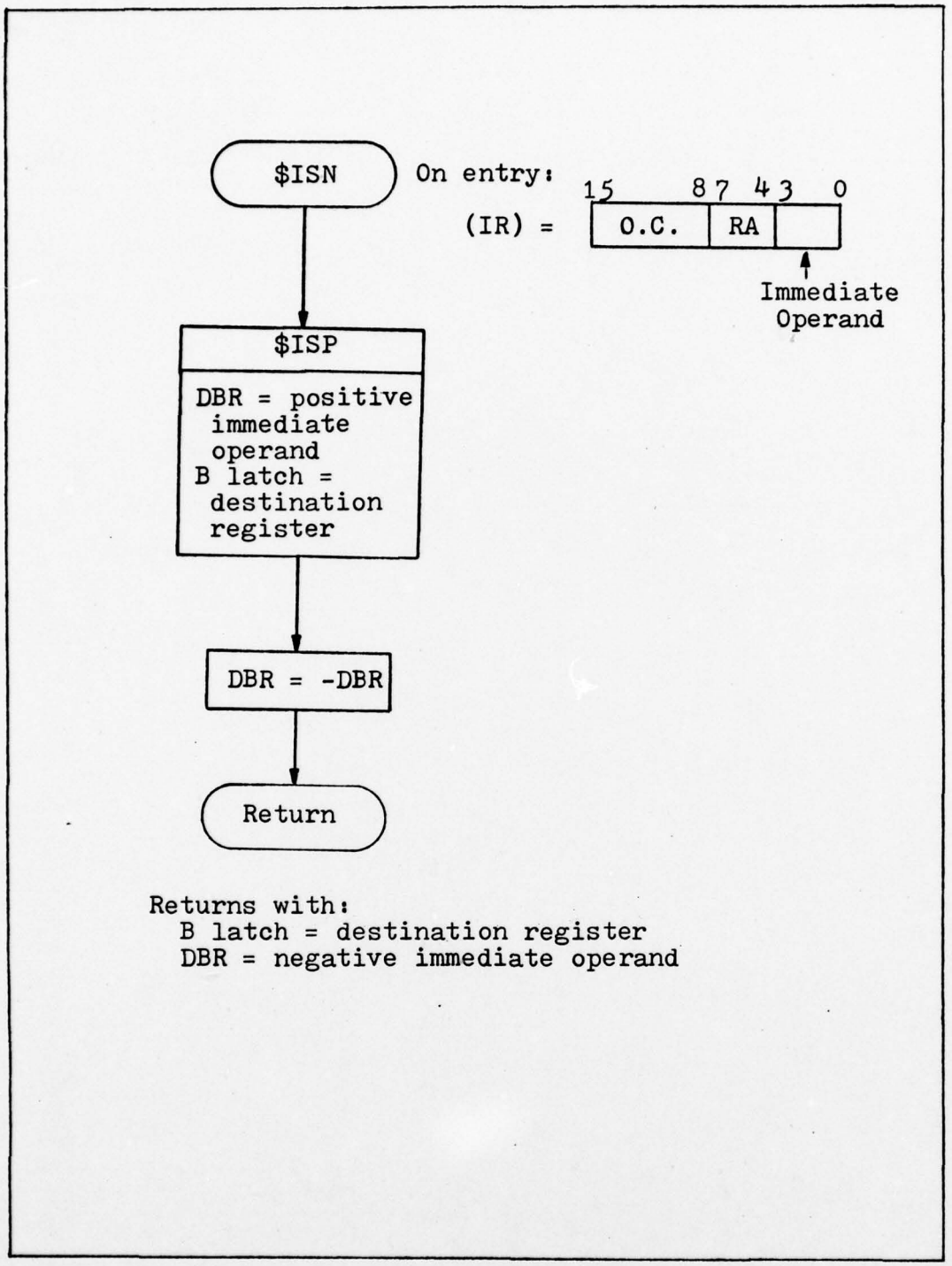


Figure B9.
Immediate Short Negative Address Mode Decoding

properly. Then the displacement is added to the current PC and stored back into the PC, so that the next instruction executed will be the one at the destination of the branch. This subroutine, therefore, is called after it has been determined that the branch will be taken (e.g., after it is determined that the condition has been met in a conditional branch instruction). The decoding of this addressing mode is contained in subroutine $ICR, as shown in Figure B10.

Base Relative Non-Indexible Addressing. In this mode, one source operand and the destination register is either register 2 for single precision instructions or register Ø for floating point instructions. The address of the other operand is computed by adding the contents of the base register (either register 4, 5, 6 or 7) to the displacement field of the instruction word (a positive number between Ø and 255_{10}). The decoding for this addressing mode is contained in subroutine $B, as shown in Figure B11.

This address decoding initially sets the A latch to the base address register. The Q register is cleared and the lower byte of the IR (the displacement) loaded into it. The MAR is then loaded with the sum of the values in the A latch register (the base address) and the Q register (the displacement).

Base Relative Indexible Addressing. In this mode, one source operand and the destination is register Ø for floating point instructions and register 2 for single precision instructions. The address of the second source

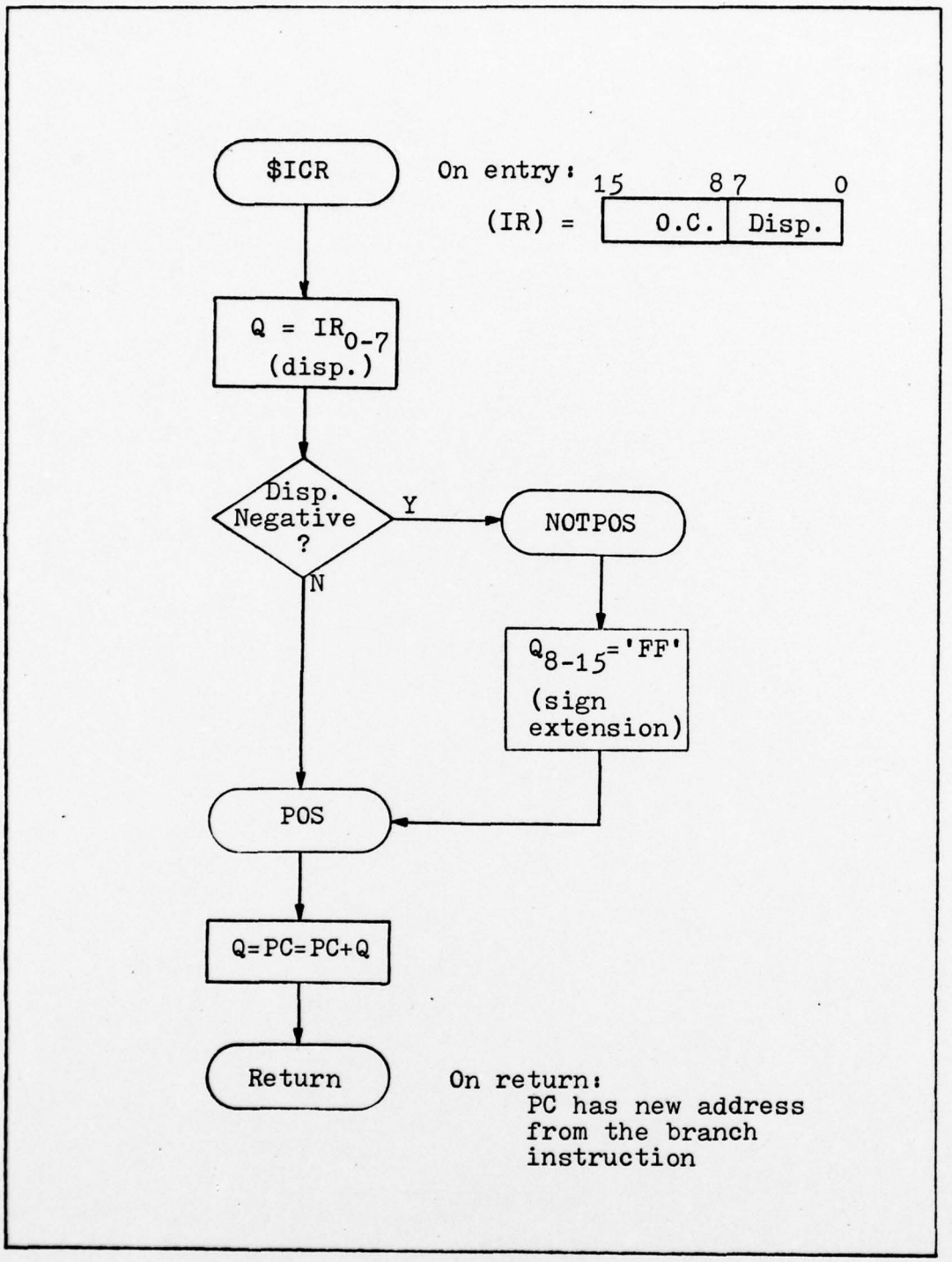


Figure B10.
Instruction Counter Relative Address Mode Decoding

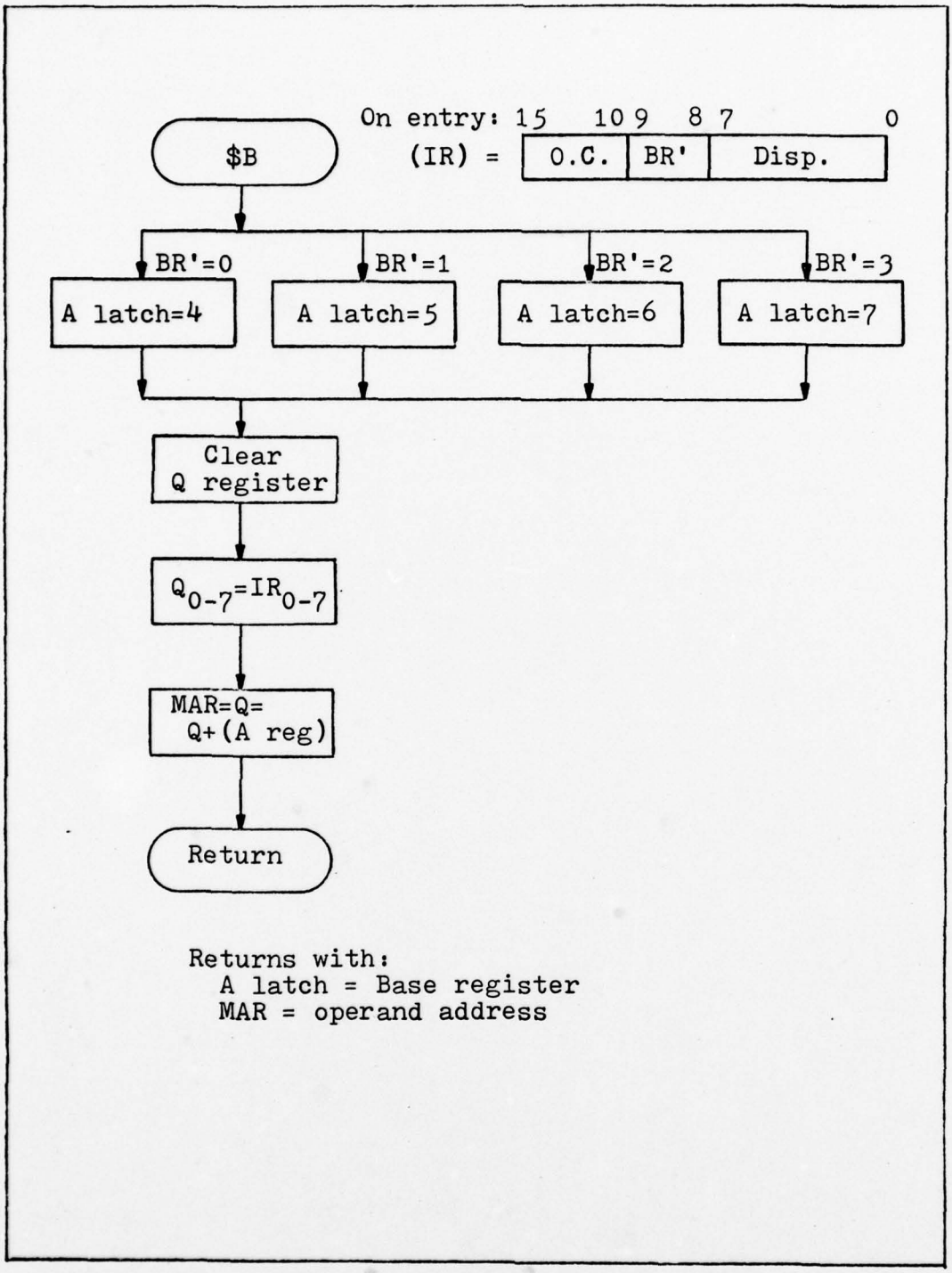


Figure B11.
Base Relative Non Indexible Address Mode Decoding

operand is derived by adding the contents of the base register (register 4, 5, 6 or 7) to the contents of an index register (register 1 through 15). An index register of zero implies no indexing.

All base relative indexible instructions using the same base register have the same opcode (40_{16}, 41_{16}, 42_{16}, or 43_{16} for base registers 4, 5, 6 or 7, respectively). An op code extension in bits 4-7 of the instruction is used to differentiate the instruction operations. The decoding begins when the mapping PROM branches to one of four addresses above, depending on which base register is specified. The B latch is set to the base register number and the Q register is cleared. Then the index register field is masked out of the IR. If this field is non-zero, the index register value is added to the base register value and the result loaded into the MAR. If it is not an indexed instruction, then the value in the base register is loaded into the MAR. The op code extension in IR_{7-4} is then used to branch to the proper routine to handle the instruction execution. This processing for address decoding is shown in figure B12.

Instruction Execution.

The microcode to execute each MIL-STD-1750 instruction is an implementation of the register transfer language contained in the standard. In order to illustrate how instructions were implemented and the use of MIME features in the implementation, several of the instructions of varying degrees of difficulty will be discussed in detail.

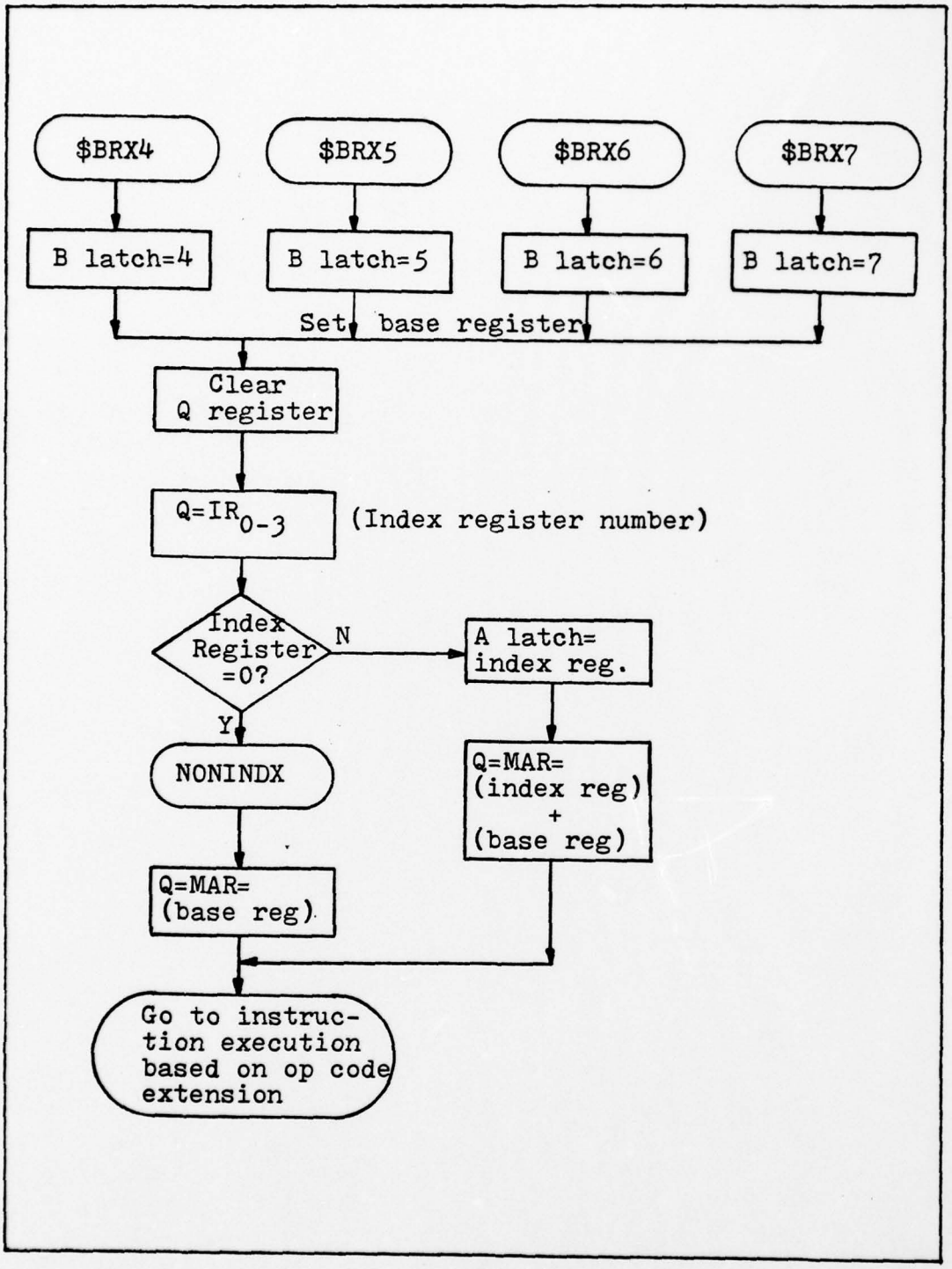


Figure B12.
Base Relative Indexible Address Mode Decoding

Single Precision Integer Add Example. This example is one of the simplest and straight-forward implementations of an instruction and is shown in Figure B13. Other than special consideration for each of the addressing modes, the general processing is to decode the operand, perform the addition, and check for overflow. Note that the entry points %AB (base relative non-indexed addressing), %A (memory direct addressing), %AISP (immediate short positive addressing) and %AR (register direct addressing) all begin with determining the operand addresses by calling a subroutine or directly setting the register address latches. %ABX (base relative indexed addressing) and $AIM (immediate long non-indexed addressing) both enter the add processing "in the middle", since these two addressing modes use opcode extensions to determine which instruction is being executed. Once the operands are determined and loaded into a standard location (in most cases, the B register latch and the DBR), processing for all addressing modes is the same.

The necessity of working around the capabilities of MIME can be seen by examining the processing following the label %AIM. At this point, the two operands for the add are in the B register and the DBR. However, this register pair is not a valid register pair for ALU operations, so that the added operation of moving the contents of the B register to the ALU's internal Q register must be performed. In the emulation, the problem of a limited number of registers and the restricted use of some registers is encountered frequently,

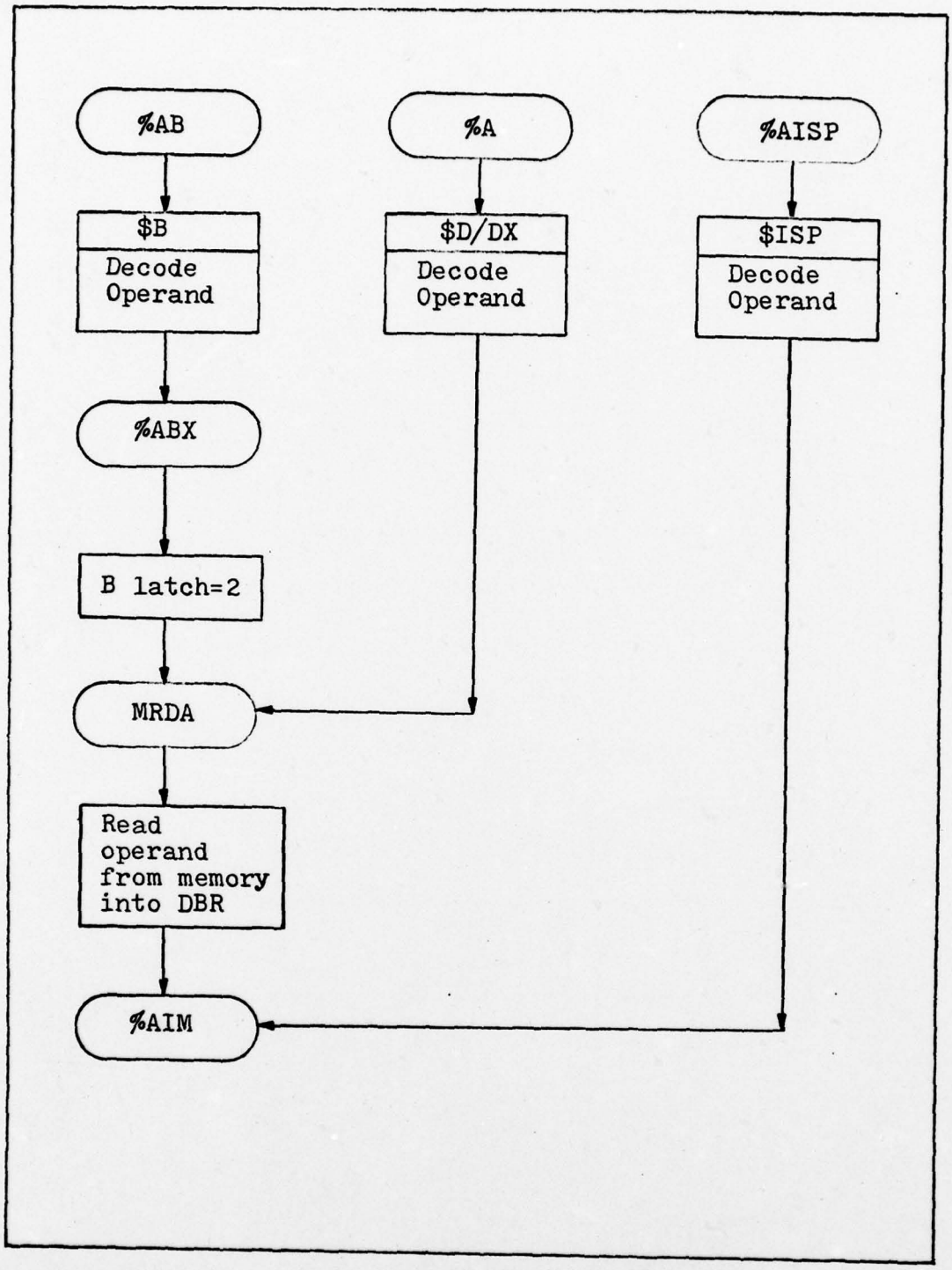


Figure B13.
Single Precision Integer Add Flow Chart

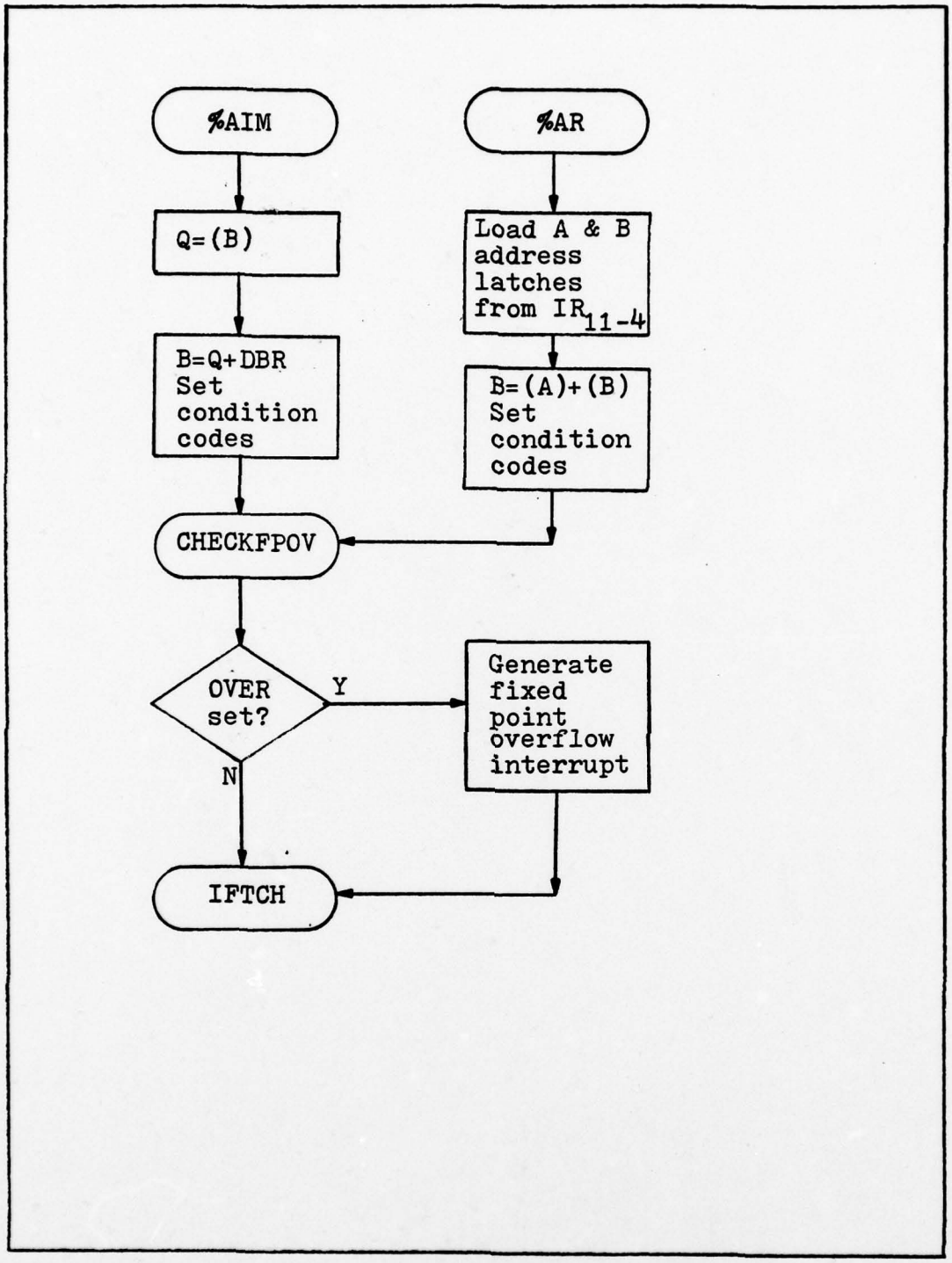


Figure B13 (continued).

necessitating a large amount of register movement.

Single Precision Integer Compare Example. This instruction is to compare the value in the RA register with the derived operand, setting the status word according to the results. Figure B14 shows the processing for this instruction.

The microcode for the integer compare starts by decoding the addressing mode and getting the operands in standard locations, as was the case for the integer add instructions. At label %CIM, the value in the B latch register (which is the RA register) is moved to the Q register, while the other operand for the comparison is in the DBR.

The subroutine CMPR is used in this and other compare instructions and is shown in figure B15. The ALU uses two's complement arithmetic, which prohibits using a plain subtraction to perform the comparison. To illustrate the problem, suppose that the RA value (now in the Q register) is a large positive number such as $7FF0_{16}$, while the other operand is a large negative number such as 8001_{16}. The correct comparison is that RA is greater than the derived operand, and both the zero and the negative MIME condition codes should be cleared (which in turn causes the positive condition in the status word to be set). However, two's complement subtraction of the two operand yields

$$7FF0_{16} - 8001_{16} =$$

$$7FF0_{16} + 7FFE_{16} + 1 = EFFF_{16}$$

which is a negative number. CMPR is designed to handle this

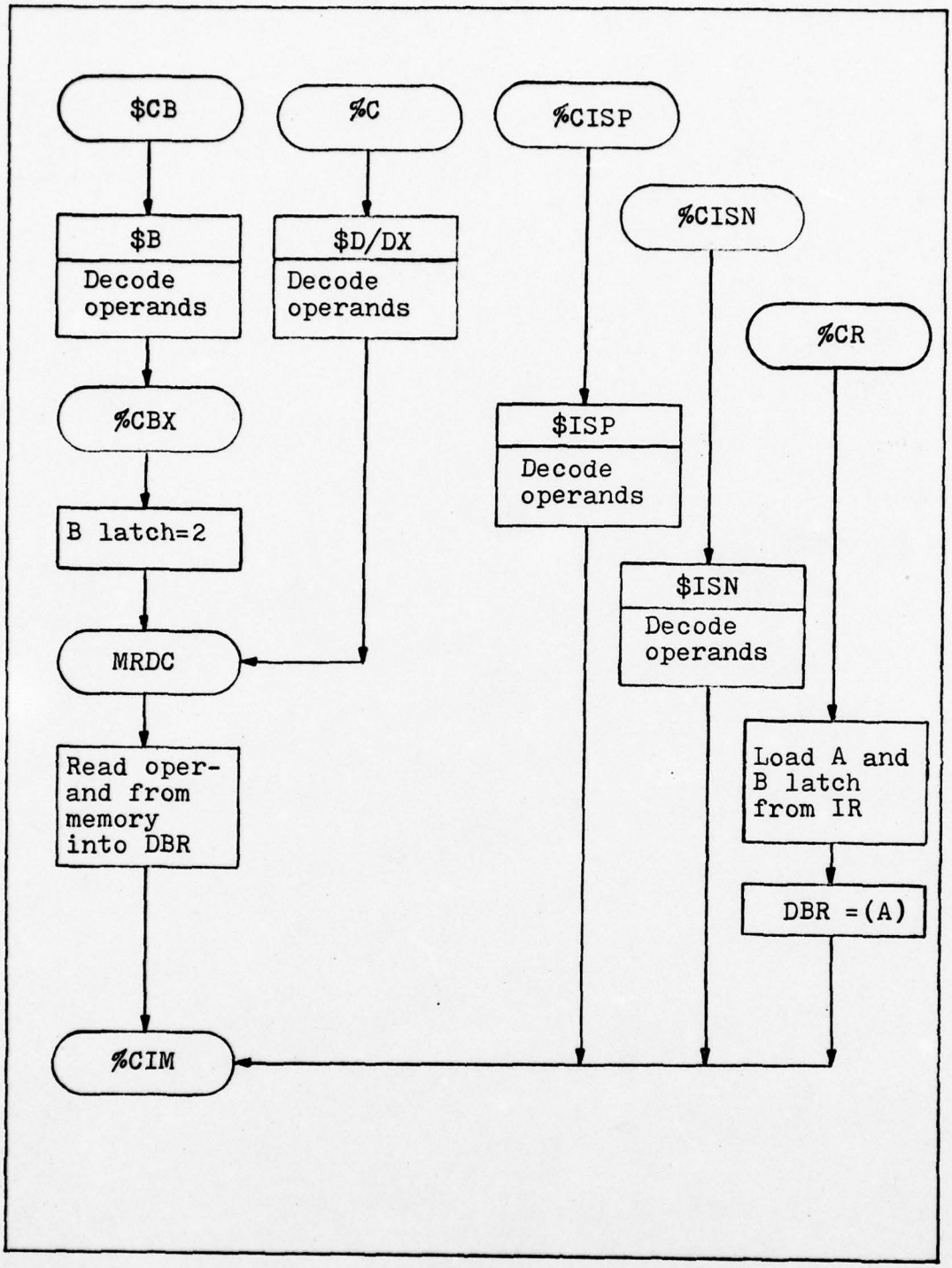


Figure B14.
Single Precision Integer Compare Flow Chart

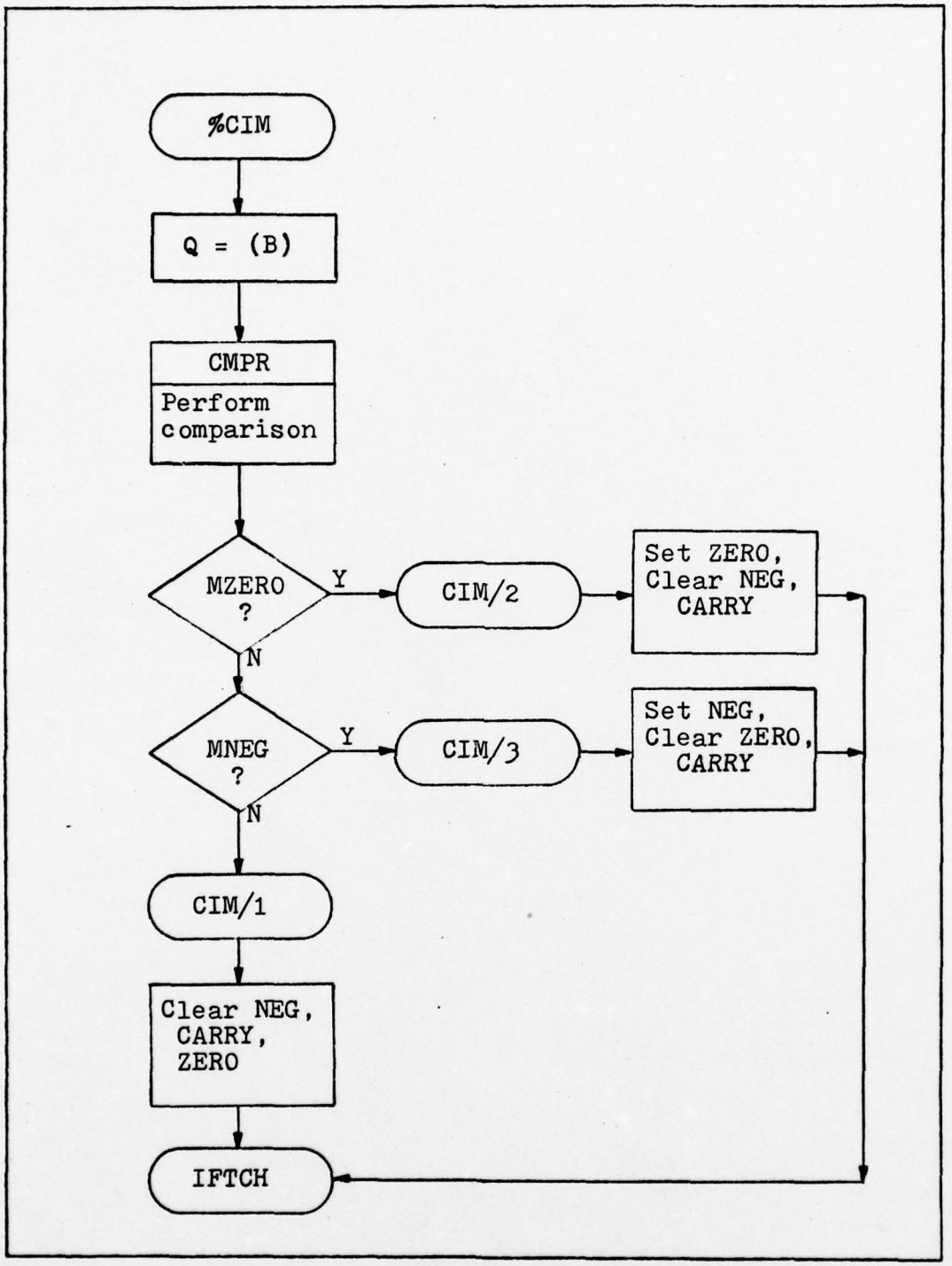


Figure B14 (continued)

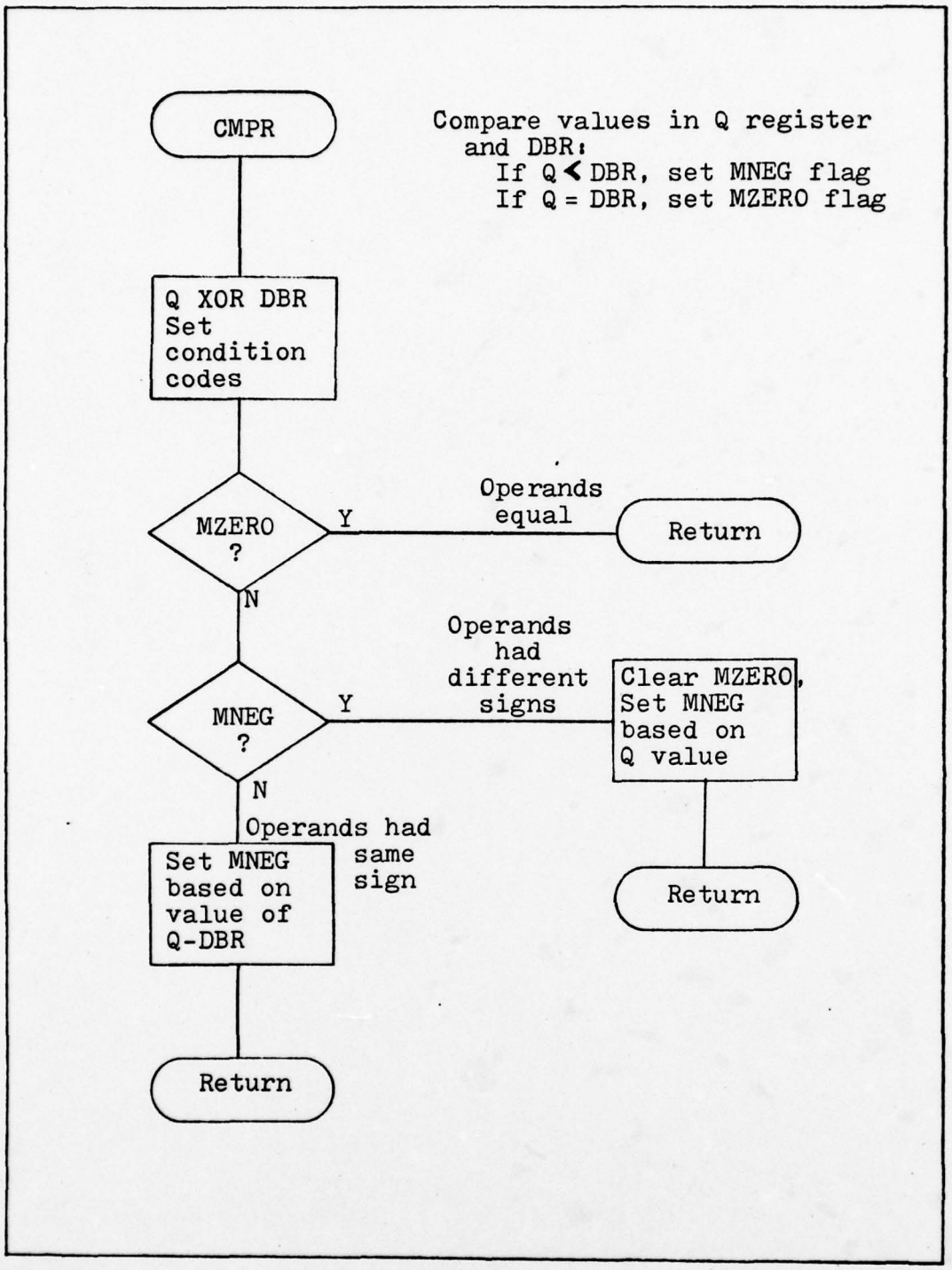


Figure B15.
CMPR Subroutine Flow Chart

type of situation. First, the subroutine checks if the operands are equal. If not, then the signs of the two operands are inspected. If the signs are the same, then a regular two's complement subtraction will accomplish the comparison. If the signs are not the same, then the larger operand will be the one that is positive. On return from CMPR, the MZERO and MNEG micro condition codes can be checked for the results of the compare and the macro condition codes set accordingly.

Floating Point Multiply Example. This represents one of the more complex MIL-STD-1750 instruction implementations. It is shown in figure B16.

The processing up to the label FMR/2 is involved with loading operands in registers. When this point is reached, AUX1 and AUX4 contain the most significant and least significant half, respectively, of the RA operand, while AUX5 and AUX6 contain the corresponding halves of the derived operand. The subroutine FLPLOADREG loads the derived operand from registers, while FLPLOADMEM loads it from memory locations.

The subroutine FLPAS/S1 separates the mantissas and exponents of the two operands in preparation for multiplication. This subroutine leaves the RA mantissa in AUX1 and AUX2, the RA exponent in AUX3 and the DBR, the derived operand exponent in AUX4 and the Q register, and the derived operand mantissa in AUX5 and AUX6.

Adding the exponents to accomplish that part of the multiplication is performed by the subroutine FLPMULEX.

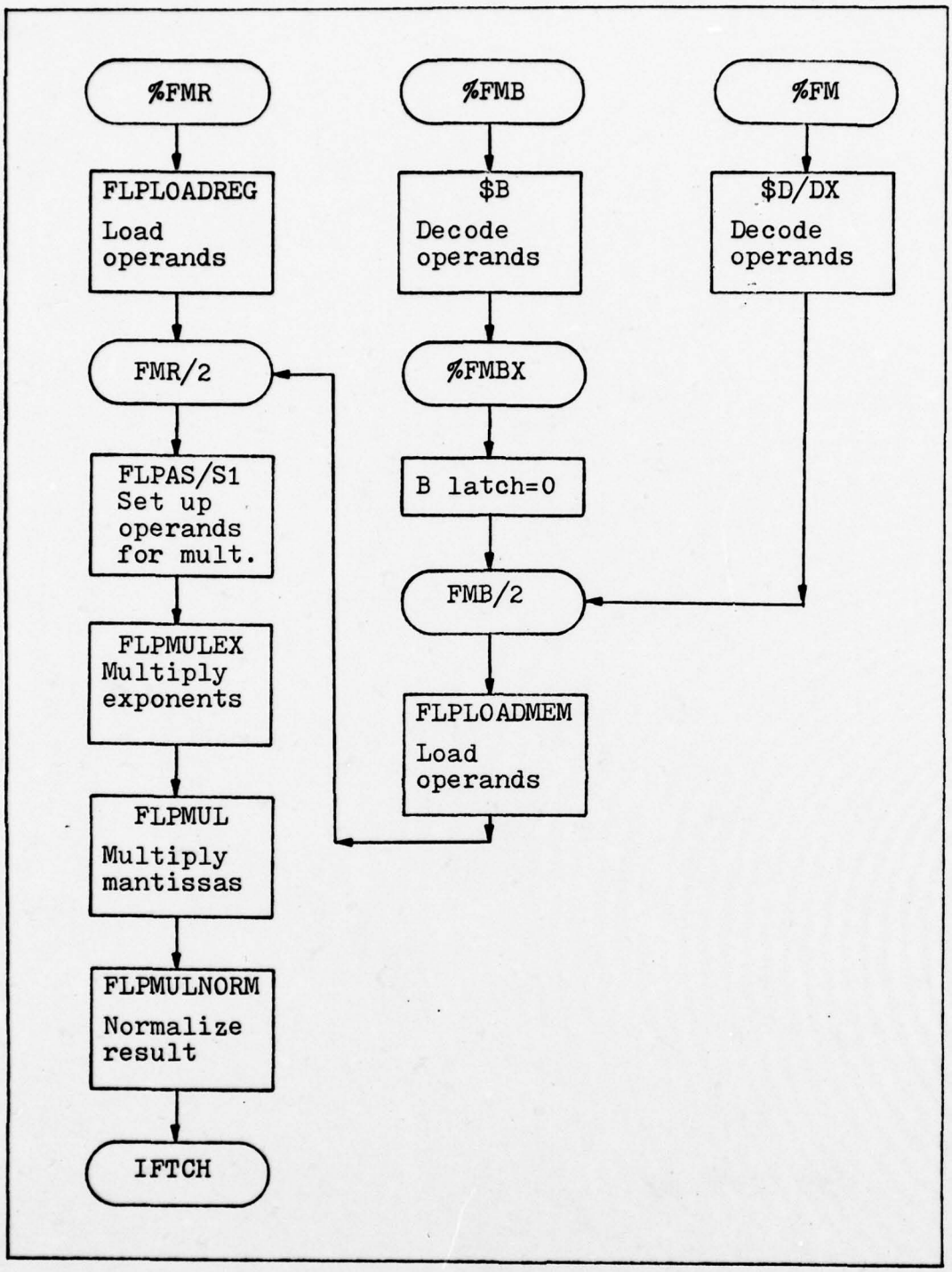


Figure B16.
Floating Point Multiply Flow Chart

The Q register and the DBR are added, with the result stored in the least significant half of the destination register. A check is made for floating point underflow or overflow as a result of this addition, and the appropriate interrupt generated if necessary.

The subroutine FLPMUL, shown in figure B17, performs the multiplication of the mantissas. This routine begins by first insuring that both operands are positive. If the result of the multiplication will have to be negated, UDTCC is set as a flag. The RA operand in AUX1 and AUX2 is moved to AUX3 and AUX4, since the algorithm used for the multiplication will do successive right shifts and adds. If the least significant bit of AUX4 is a one, the derived operand in AUX5 and AUX6 will be added to AUX1 and AUX2. Then AUX1 through AUX4 will be right shifted and the process repeated 32 times. The result is then in AUX1 through AUX4.

The subroutine FLPMULNORM normalizes the result. The result mantissa is put into AUX1 and AUX2 (changing its sign if UDTCC is set) and the exponent into AUX5. FLPNORMALIZE, as shown in figure B18, is called to shift the result until the sign bit of the mantissa has a value opposite that of the next most significant bit. The subroutine also checks for underflow during the normalization process and generates the floating point underflow interrupt, if necessary. Finally, the result is loaded into the destination registers.

Interrupt Handling.

The step prior to the instruction fetch processing is to

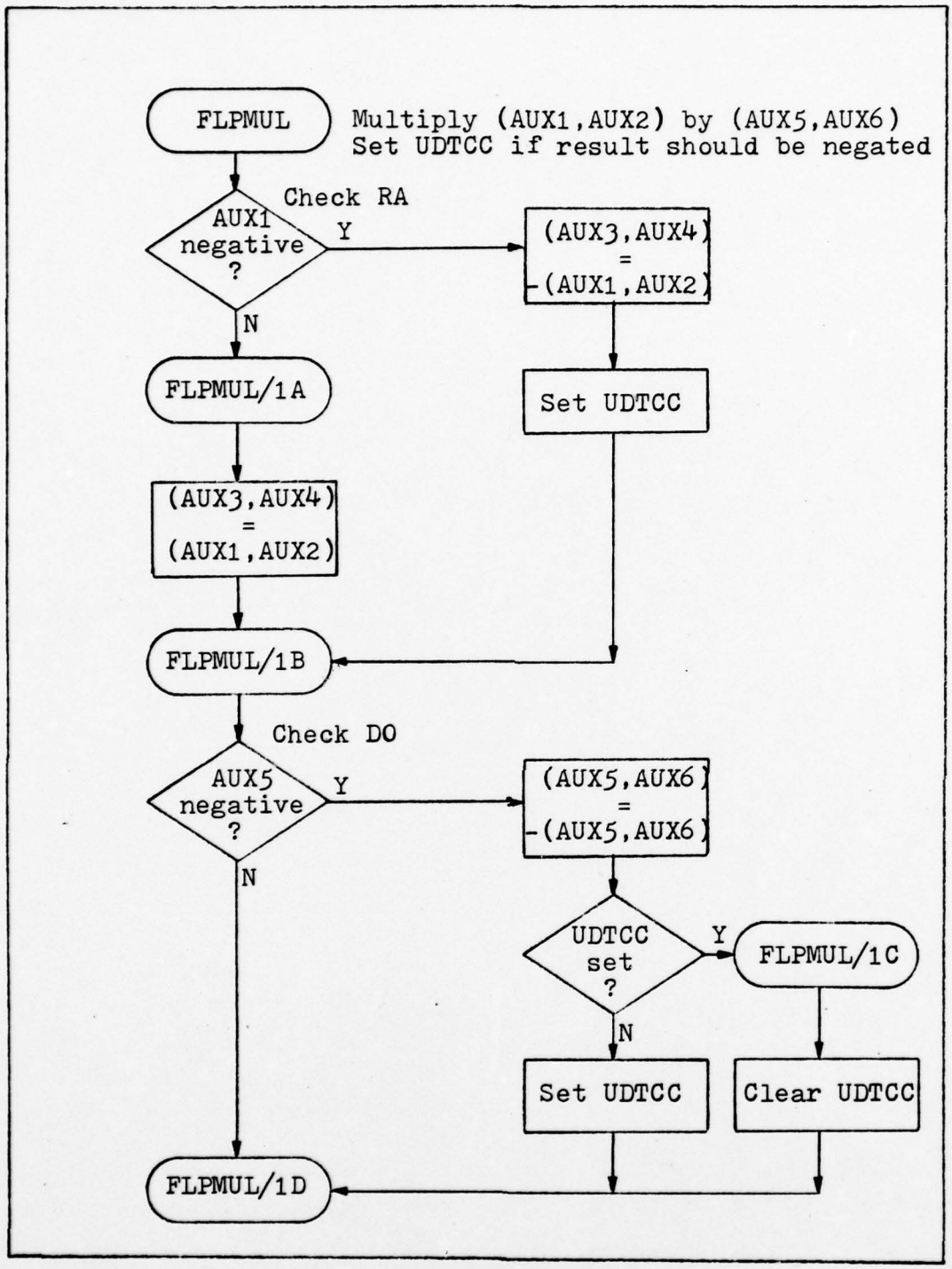


Figure B17.
FLPMUL Subroutine Flow Chart

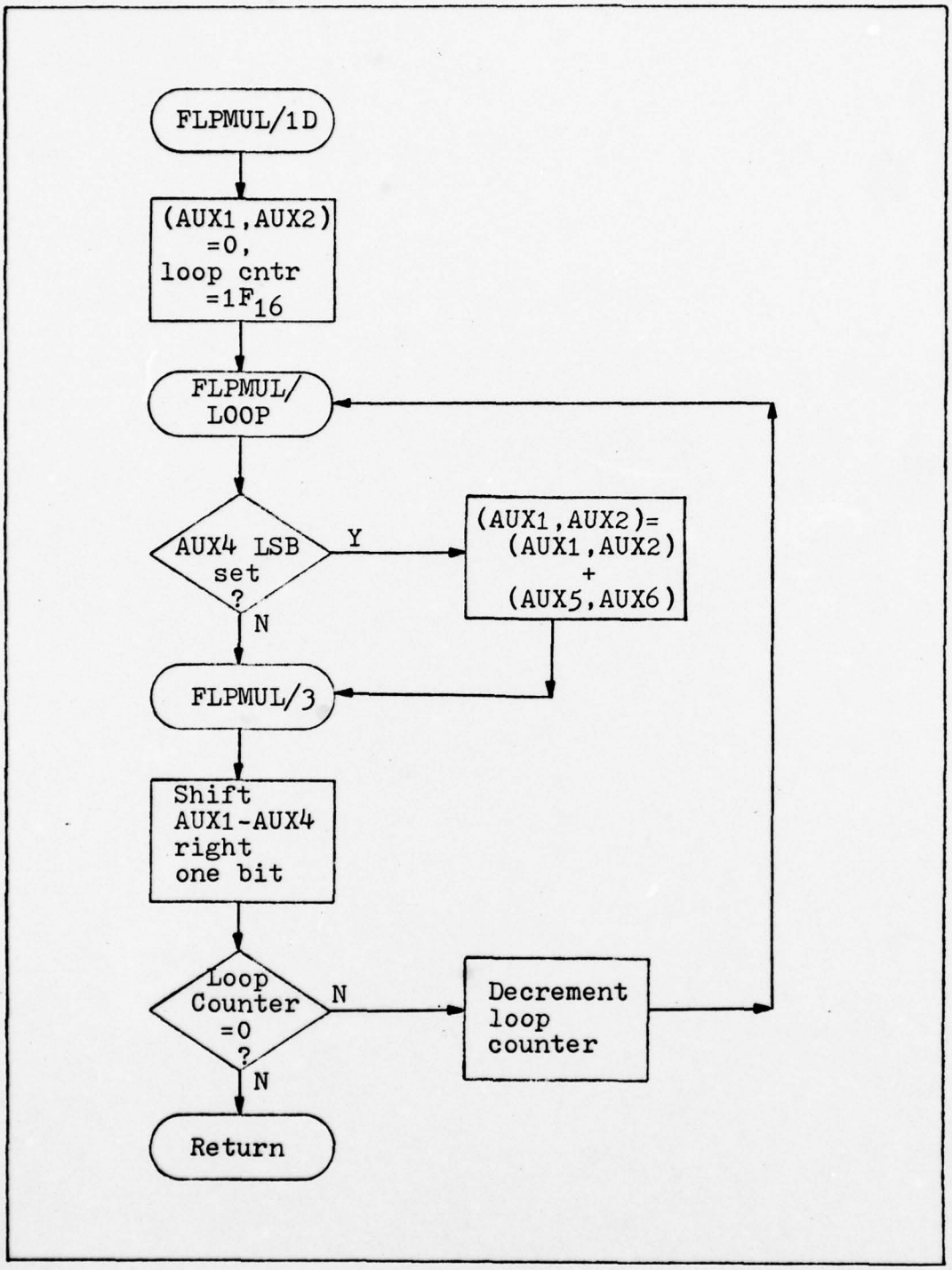


Figure B17 (continued)

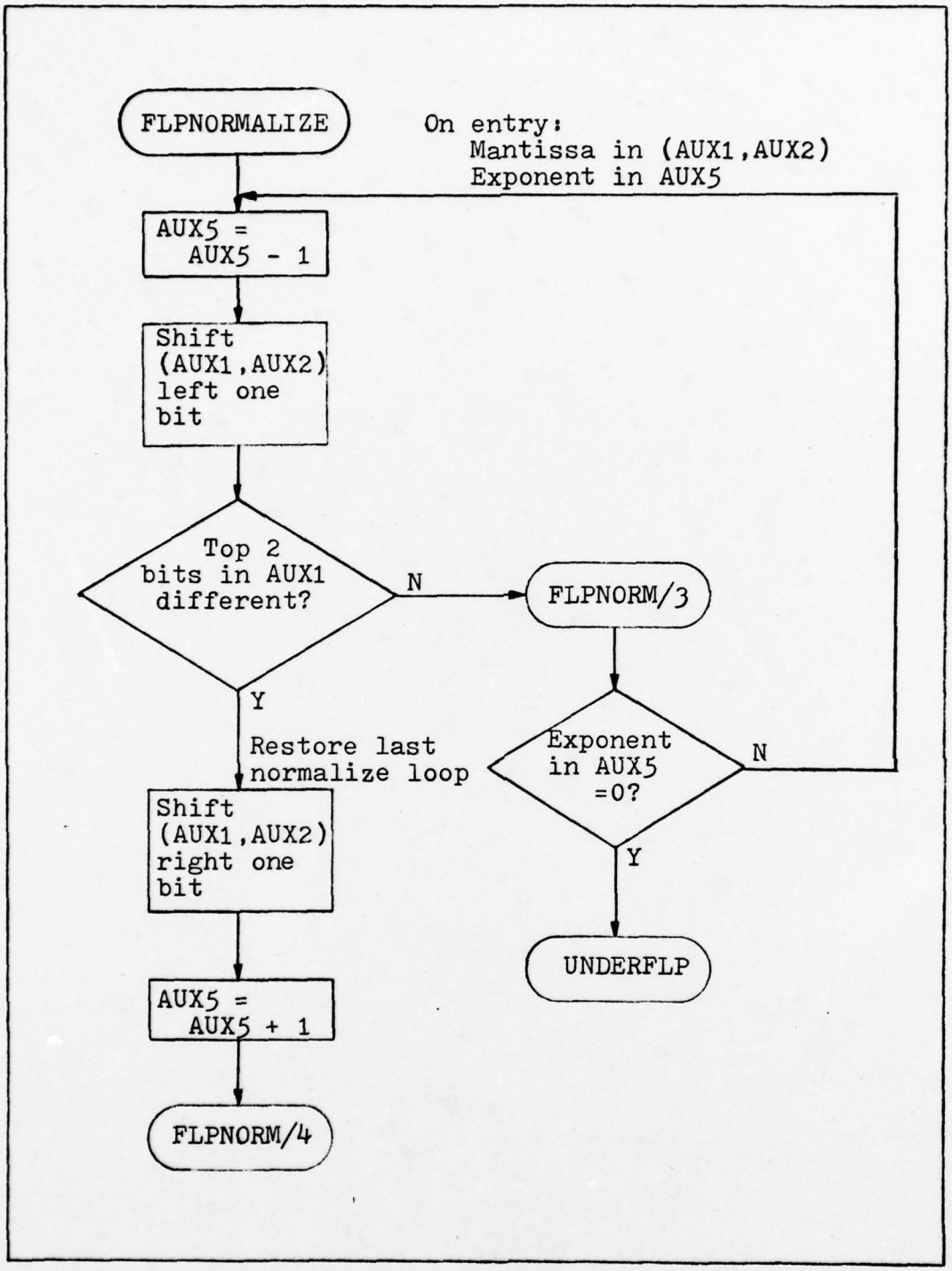


Figure B18.
FLPNORMALIZE Subroutine Flow Chart

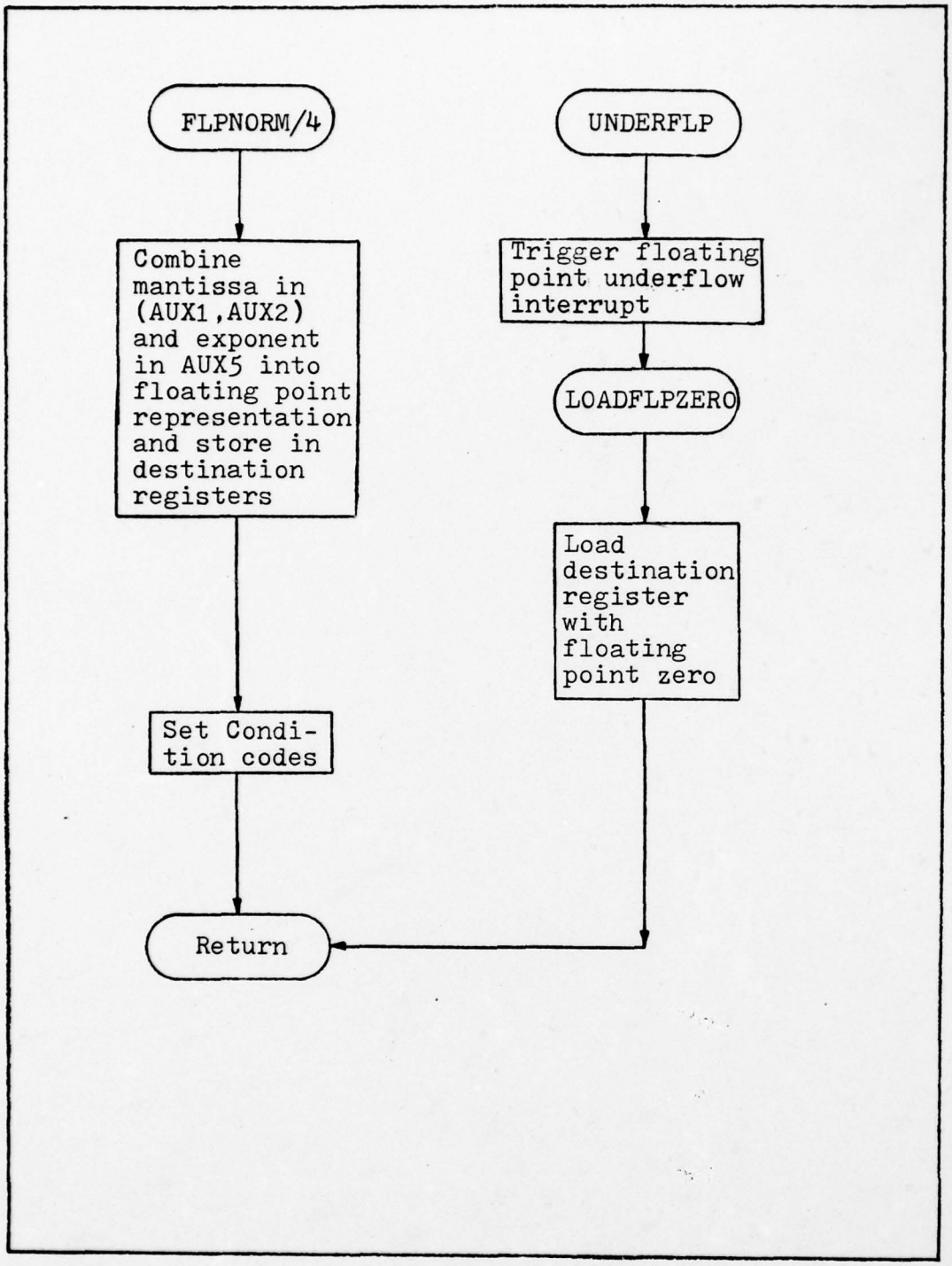


Figure B18 (continued)

check if an interrupt is pending. If so, a branch to the interrupt handler routine, INTHNDL, is taken.

The interrupt system defined in MIL-STD-1750 (Ref. 2 :15) is based on a vector table of interrupt linkage and service pointers. Each interrupt has its own linkage pointer and service pointer in the table. A linkage pointer points to three words of memory where the interrupt mask, status word and program counter are stored when that interrupt occurs. The service pointer points to three words of memory where new values for the interrupt mask, status word and program counter are to be found. These values are loaded when that interrupt occurs to start the service routine processing.

INTHNDL, as shown in figure B19, first determines which of the sixteen interrupts has occurred by examining the interrupt vector and transferring to one of the sixteen interrupt routines. The interrupt routine clears this interrupt from the interrupt control unit, saves the current interrupt mask, status word and program counter in the interrupt linkage area, and loads the new values of these registers from the interrupt service area. The new PC, loaded from the interrupt service area, determines the location of the user's interrupt service routine.

The interrupt vector table is specified to start at memory address 20_{16}. Table BV defines the locations of the interrupt linkage and service pointers in this table.

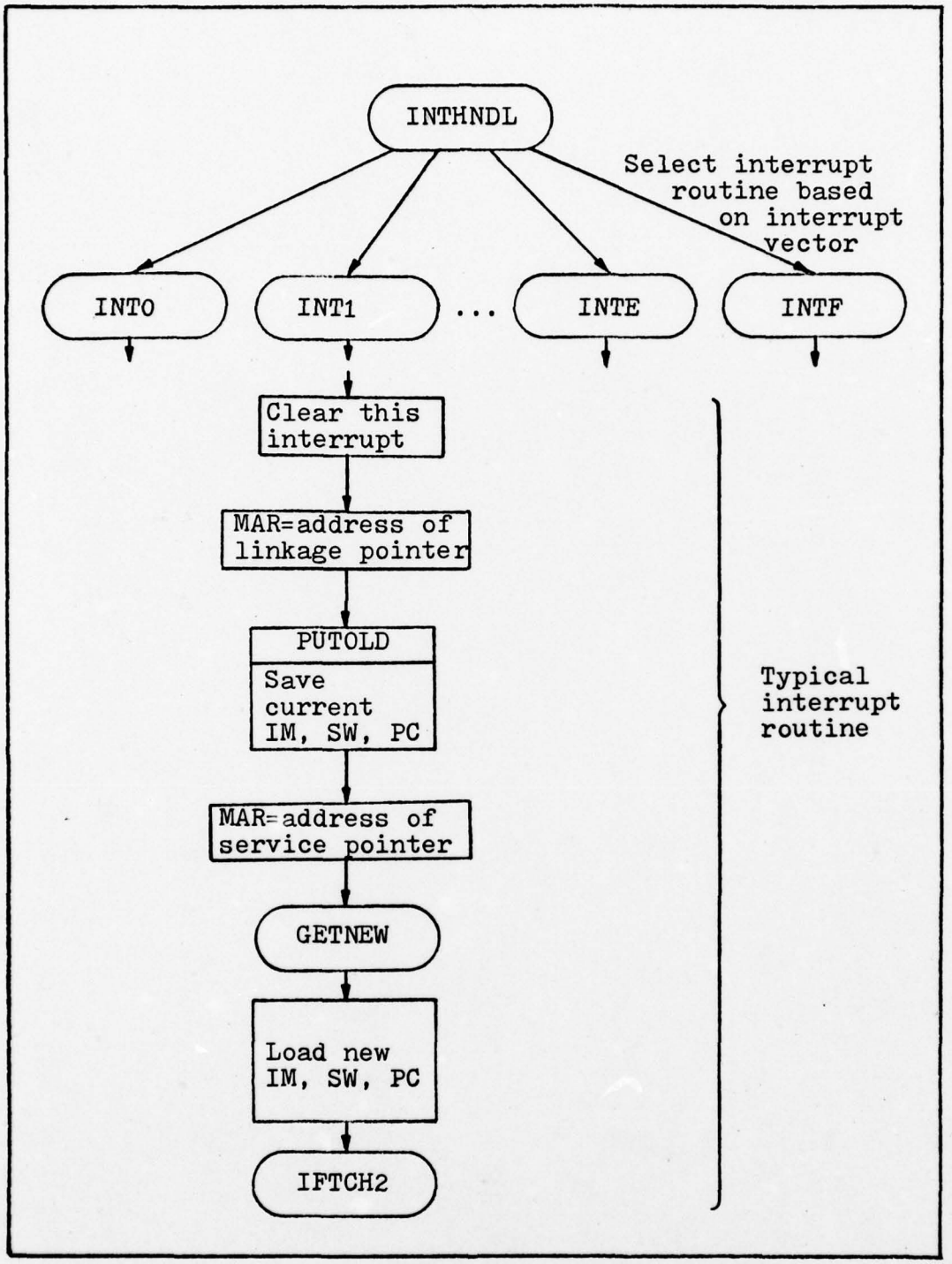


Figure B19.
Interrupt Handler Processing

Table BV

Interrupt Vector Table

Memory Location (base 16)		Contents
2Ø	Highest	Interrupt Ø linkage pointer
21	Priority	Interrupt Ø service pointer
22		Interrput 1 linkage pointer
23		Interrupt 1 service pointer
24		Interrupt 2 linkage pointer
25		Interrupt 2 service pointer
26		Interrupt 3 linkage pointer
27		Interrupt 3 service pointer
28		Interrupt 4 linkage pointer
29		Interrupt 4 service pointer
2A		Interrupt 5 linkage pointer
2B		Interrupt 5 service pointer
2C		Interrupt 6 linkage pointer
2D		Interrupt 6 service pointer
2E		Interrupt 7 linkage pointer
2F		Interrupt 7 service pointer
3Ø		Interrupt 8 linkage pointer
31		Interrupt 8 service pointer
32		Interrupt 9 linkage pointer
33		Interrupt 9 service pointer
34		Interrupt 10 linkage pointer
35		Interrupt 10 service pointer
36		Interrupt 11 linkage pointer
37		Interrupt 11 service pointer
38		Interrupt 12 linkage pointer
39		Interrupt 12 service pointer
3A		Interrupt 13 linkage pointer
3B		Interrupt 13 service pointer
3C		Interrupt 14 linkage pointer
3D		Interrupt 14 service pointer
3E	Lowest	Interrupt 15 linkage pointer
3F	Priority	Interrupt 15 service pointer

Microprogram Performance

Each MIL-STD-1750 instruction follows a particular sequence of microinstructions to execute. By counting the microinstructions involved in each MIL-STD-1750 instruction execution, the performance of the emulation can be determined. Table BVI contains the maximum number of microinstructions that will be executed for each MIL-STD-1750 instruction. Execution time for each instruction can be determined by multiplying this count by the period of the MIME clock.

Table BVI. MIL-STD-1750 Emulation Performance Statistics

Mnemonic	Microinstruction Count[1]	Instruction
<u>Overhead</u>		
IFTCH	5	Instruction fetch
INTHNDL	24	Interrupt initiation
D	10	Memory direct address decoding
DX	13	Memory direct indexed address decoding
I	13	Memory indirect address decoding
IX	16	Memory indirect indexed address decoding
IM	7	Immediate long indexible address decoding(without indexing)
IMX	10	Immediate long indexible address decoding(with indexing)
IMML	6	Immediate long non-indexible address decoding
ISP	6	Immediate short positive address decoding
ISN	9	Immediate short negative address decoding
ICR	7 (displacement≥0)	Instruction counter relative address decoding
	9 (displacement<0)	
B	6	Base relative non-indexible address decoding
BX	9 (no indexing) 11 (indexing)	Base relative indexible address decoding
R	0	Register direct address decoding

Note 1 - Total microinstructions required to execute a MIL-STD-1750 instruction consists of the sum of the microinstruction counts for IFTCH, address decoding, and instruction execution.

Table BVI (continued)

Instruction Execution		
A	7	Single precision integer add
AB	8	Single precision integer add
ABS	4/10[2]	Single precision absolute value of register
ABX	7	Single precision integer add
AND	5	Logical AND
ANDB	6	Logical AND
ANDM	2	Logical AND
ANDR	2	Logical AND
ANDX	5	Logical AND
AIM	4	Single precision add
AISP	6	Single precision add
AR	4	Single precision add
BEZ	4/7[3]	Branch if equal to (zero)
BGE	4/7[3]	Branch if greater than or equal to (zero)
BGT	4/7[3]	Branch if greater than (zero)
BLE	4/7[3]	Branch if less than or equal to (zero)
BLT	4/7[3]	Branch if less than (zero)
BNZ	4/7[3]	Branch if not equal to (zero)
BR	2	Branch unconditionally
C	13	Single precision compare
CB	14	Single precision compare
CBL	23	Compare between limits
CBX	13	Single precision compare
CIM	10	Single precision compare

Note 2 - Four microinstructions are need if the value does not need to be negated. Ten are needed if the value has to be negated.

Note 3 - If the branch is not taken, only 4 microinstructions are needed. If the branch is taken, then 7 microinstruction plus the ICR address decoding count is consumed.

Table BVI (continued)

CISN	12	Single precision compare
CISP	12	Single precision compare
CR	13	Single precision compare
D	425	Single precision integer divide (32 bits)
DA	29	Double precision integer add
DABS	18	Double precision absolute value of register
DAR	32	Double precision integer add
DB	427	Single precision integer divide (32 bits)
DBX	426	Single precision integer divide (32 bits)
DC	22	Double precision compare
DCR	25	Double precision compare
DD	832	Double precision integer divide
DDR	835	Double precision integer divide
DECM	17	Decrement memory by a positive integer
DIM	423	Single precision integer divide (32 bit dividend)
DISN	418	Single precision integer divide (16 bit dividend)
DISP	418	Single precision integer divide (16 bit dividend)
DL	12	Double precision load
DLB	12	Double precision load
DLBX	11	Double precision load
DLI	11	Double precision load
DLR	9	Double precision load
DM	308	Double precision integer multiply
DMR	311	Double precision integer multiply
DNEG	19	Double precision negate register
DR	425	Single precision integer divide (32 bit dividend)
DS	30	Double precision integer subtract
DSAR	19+4N	Double shift arithmetic, count in register
DSCR	19+2N	Double shift cyclic, count in register
DSLC	26+2N	Double shift left cyclic
DSLL	26+2N	Double shift left logical
DSLR	19+2N	Double shift logical, count in register

N = number of shifts to perform

Table BVI (continued)

DSR	33	Double precision integer subtract
DSRA	26+2N	Double shift right arithmetic
DSRL	26+2N	Double shift right logical
DST	7	Double precision store
DSTB	7	Double precision store
DSTI	7	Double precision store
DSTX	6	Double precision store
DV	419	Single precision integer divide (16 bit dividend)
DVIM	417	Single precision integer divide (16 bit dividend)
DVR	420	Single precision integer divide (16 bit dividend)
EFA	N/I	Extended precision floating point add
EFAR	N/I	Extended precision floating point add
EFC	N/I	Extended precision floating point compare
EFCR	N/I	Extended precision floating point compare
EFD	N/I	Extended precision floating point divide
EFDR	N/I	Extended precision floating point divide
EFIX	N/I	Convert extended precision floating point to 32 bit integer
EFL	19	Extended precision floating point load
EFLT	N/I	Convert 32 bit integer to extended precision floating point
EFM	N/I	Extended precision floating point multiply
EFMR	N/I	Extended precision floating point multiply
EFS	N/I	Extended precision floating point subtract
EFSR	N/I	Extended precision floating point subtract
EFST	9	Extended precision floating point store
FA	443	Floating point add
FAB	443	Floating point add
FABS	38	Floating point absolute value of register
FABX	442	Floating point add
FAR	444	Floating point add
FC	36	Floating point compare
FCB	36	Floating point compare
FCBX	35	Floating point compare
FCR	38	Floating point compare
FD	1192	Floating point divide
FDB	1190	Floating point divide
FDBX	1189	Floating point divide

N/I = instructions that were not implemented in this emulation

Table BVI (continued)

FDR	1192	Floating point divide
FIX	72	Convert floating point to 16 bit integer
FLT	336	Convert integer to floating point
FM	681	Floating point multiply
FMB	681	Floating point multiply
FMBX	680	Floating point multiply
FMR	683	Floating point multiply
FNEG	41	Floating point negate
FS	447	Floating point subtract
FSB	447	Floating point subtract
FSBX	446	Floating point subtract
FSR	448	Floating point subtract
IN	24	Input
INCM	16	Increment memory by a positive integer
JC	8/10[4]	Jump on condition
JCI	8/10[4]	Jump on condition
JS	4	Jump to subroutine
L	5	Single precision load
LB	5	Single precision load
LBX	4	Single precision load
LDST	9	Load status
LI	4	Single precision load
LIM	2	Single precision load
LISN	2	Single precision load
LISP	2	Single precision load
LLB	6	Load from lower byte
LLBI	5	Load from lower byte
LM	14+4R	Load multiple registers
LR	2	Single precision load

Note 4 - Eight microinstructions are needed if the jump does not occur. If the condition is met and the jump occurs, 10 microinstructions plus the address decoding are required.

R = number of registers involved in the instruction

Table BVI (continued)

LUB	24	Load from upper byte
LUBI	23	Load from upper byte
M	104	Single precision integer multiply (32 bit produst)
MB	104	Single precision integer multiply (32 bit product)
MBX	103	Single precision integer multiply (32 bit product)
MIM	100	Single precision integer multiply (32 bit product)
MISM	103	Single precision integer multiply (16 bit product)
MISN	104	Single precision integer multiply (16 bit product)
MISP	104	Single precision integer multiply (16 bit product)
MOV	7+21W	Move multiple words
MR	108	Single precision integer multiply (32 bit product)
MSR	104	Single precision integer multiply (16 bit product)
N	6	Logical NAND
NEG	9	Single precision negate register
NIM	3	Logical NAND
NOP	1	No operator
NR	3	Logical NAND
OR	5	Inclusive logical OR
ORB	6	Inclusive logical OR
ORBX	5	Inclusive logical OR
ORIM	2	Inclusive logical OR
ORR	2	Inclusive logical OR
OUT	46	Output
POPM	25+6R	Pop multiple registers off stack
PSHM	22+3R	Push multiple registers onto stack
RB	25+2(B+1)	Reset bit
RBI	26+2(B+1)	Reset bit
RBR	29+2(B+1)	Reset bit
RVBR	32+2(B+1)	Reset variable bit in register
S	7	Single precision integer subtract

W = number of words to move
R = number of registers involved in this instruction
B = bit number, where the most significant bit is bit number 0

Table BVI (continued)

SAR	17+4N	Shift arithmetic, count in register
SB	24+2(B+1)	Set bit
SBB	8	Single precision integer subtract
SBBX	7	Single precision integer subtract
SBI	25+2(B+1)	Set bit
SBR	28+2(B+1)	Set bit
SCR	16+2N	Shift cyclic, count in register
SIM	4	Single precision integer subtract
SISP	6	Single precision integer subtract
SJS	5	Stack IC and jump to subroutine
SLBI	3	Store into lower byte
SLC	23+2N	Shift left cyclic
SLL	22+2N	Shift left logical
SLR	16+2N	Shift logical, count in register
SOJ	3/5[5]	Subtract one and jump
SR	4	Single precision integer subtract
SRA	24+2N	Shift right arithmetic
SRL	23+2N	Shift right logical
SRM	19	Store register through mask
ST	4	Single precision store
STB	4	Single precision store
STBX	3	Single precision store
STC	15	Store a non-negative constant
STCI	14	Store a non-negative constant
STI	4	Single precision store
STM	14+3R	Store multiple registers
STLB	4	Store into lower byte
STUB	26	Store into upper byte
SUBI	25	Store into upper byte

Note 5 - Three microinstructions are needed if the result is not zero. If the result is zero and the jump occurs, five microinstructions plus the address decoding are required.

N = number of shifts to perform
R = number of registers involved in the instruction
B = bit number, where the most significant bit is bit number 0

Table BVI (continued)

SVBR	31+2(B+1)	Set variable bit in register
TB	33+2(B+1)	Test and set bit
TBI	32+2(B+1)	Test and set bit
TBR	30+2(B+1)	Test and set bit
TSB	33+2(B+1)	Test and set bit
TVBR	32+2(B+1)	Test variable bit in register
URS	5	Unstack IC and return from subroutine
XBR	19	Exchange bytes in register
XOR	5	Exclusive logical OR
XORM	2	Exclusive logical OR
XORR	2	Exclusive logical OR
XWR	11	Exchange words in register

B = bit number, where the most significant bit is bit number 0

References

1. Hoyt, Thomas R. and Dean A. Myers. MIME: Microprogrammable Minicomputer Emulator, Phase II. Unpublished Thesis. Wright-Patterson AFB: Air Force Institute of Technology, 1979.

2. MIL-STD-1750 (USAF). Airborne Computer Instruction Set Architecture. Washington: Department of Defense, 21 February 1979.

Appendix C

MIME Translator Reference Manual

Contents

List of Tables

List of Figures

I. Introduction

This appendix describes the MIME translator, a Pascal program that translates a mnemonic description of the microprogram into microcode for loading into the MIME control store. This frees the user from the error prone process of generating the bits by hand for the 64-bit microword. Throughout this manual, it is assumed that the user understands the architecture of the MIME and the capabilities of its microprogramming (see Appendix A, MIME User's Manual).

The MIME microword consists of twenty interrelated fields ranging in size from one to four bits. In order to describe the various options available in these fields, the translator recognizes a large number of reserved words. The translator also incorporates a facility to redefine a reserved word with another mnemonic, allowing the programmer to develop substitute reserved words that have more meaning in the particular microprogram or emulation being performed.

In Chapter II of this appendix, the structure of a program and its component parts is described. Chapter III describes the process of redefining a reserved word, while Chapter IV covers the possible types of microstatements that can make up a microprogram. Chapter V defines the pseudo-operations, or directives to the translator. The use of the translator on the ASD CDC computer system is covered in Chapter VI.

II. Structures

Names

A name consists of a series of letters, numbers, or special characters that is preceeded and followed by one of the set of delimiters. The length of the name is limited to the boundaries of the eighty column input card, and it must be unique in the first ten characters.

Delimiters

A delimiter is a special character that serves to define the boundaries of a name. The set of special characters that are delimiters consists of the blank, comma, colon, period, semicolon, the pound symbol (#), the plus sign, and the minus sign. These (except for the blank) are considered to be part of the set of reserved words. However, because of the unique nature of this set of characters as separators in the language, it is illegal to redefine a delimiter.

Labels

A label consists of a name followed by a colon. This may optionally be followed by a label type. If a micro-instruction is to be labeled, the label must be the first item encountered when processing a new microword in the program section. The label type is required only if the label is to be used as a multi-way branch (in which case the translator needs to place the label at a certain address) or as

Table CI

Label Types

Label Type	Meaning	Action
2	Label to be used for 2 way branch	Increment micro PC to next address divisible by 2.
4	Label to be used for 4 way branch	Increment micro PC to next address divisible by 4.
8	Label to be used for 8 way branch	Increment micro PC to next address divisible by 8.
16	Label to be used for 16 way branch	Increment micro PC to next address divisible by 16.
LOOP	Label to be used for LOOP AT FILE microstatement	Generate a PUSH FILE sequencer instruction

the destination of a 'LOOP AT FILE' statement (in which case the translator must generate an instruction to push the current address onto the stack). Table CI lists the label types.

Constants

Constants can be used as sources for information on the data bus and to set the Micro Loop Counter. Constants are signified by a string of characters delimited by apostrophies. All constants are in hexadecimal notation and will be truncated to the proper length to fit in the applicable microword field.

Reserved Words

Certain names are used to describe the microprogram.

These reserved words cause specific actions to be taken by the translator. The reserved words and their uses are described in Attachment C-A. Most of the reserved words can be renamed in the microprogram, so that the user can rename them consistent with the particular program or emulation being performed. When a reserved word is mentioned in this manual (signified by all capital letters), it is understood to mean the reserved word itself or its redefined name. When a reserved word is redefined, the original reserved word ceases to be recognized as a reserved word.

Comments

Comments consist of a group of characters surrounded by the comment character (a '#'). Comments can be placed in one of two places:

- After the period that terminates a microinstruction or a redefinition.
- On a separate card or cards with no microinstruction or redefinition preceeding the comment.

Pseudo-Operations

Pseudo-operations are directives to the translator. These must be on a card by themselves and are not terminated by a period, as are the microinstructions and redefinitions.

Program

A microprogram consists of a redefinition section, followed by the reserved word PROGRAM, followed by a program section. The redefinition section consists of any user redefinitions

of the reserved words and declarations of labels that are external to this translation. The program section consists of a sequence of microstatements which make up the microinstructions. Each microstatement typically describes the setting of one field in the microword and is followed by a semicolon, while a period follows the group of microstatements that make up a microinstruction. A comment may either follow the end of the microinstruction (the period) or may be on cards by itself. A microinstruction and its comment may continue on as many eighty column cards as necessary. However, a name may not span the card boundary. Each new microinstruction must start on a new card, and each pseudo-operation must be on a separate card. The last name in the program section must be the reserved word END, which can have no comments after it.

III. Redefinitions

A redefinition is described by the following syntax:

<new name> = <old name>

Refer to Attachment C-D for a description of the syntax format.

If <old name> is one of the original redefinable reserved words (see Attachment C-A), then <new name> is the name that will become the new redefined name of the reserved word. Once a reserved word has been redefined, the <old name> is no longer a valid reserved word and has no special meaning to the translator. As such, it can then be used as the <new name> in a succeeding redefinition.

If <old name> is a constant, then <new name> is interpreted to be a label external to this particular translation, with the constant as the label's microaddress. No label type may be attached to the label, but the label will satisfy all the checks that the translator makes for label type consistency.

If <new name> is the word 'UNUSED' (a reserved word that is not redefinable), the reserved word signified by <old name> is made an invalid reserved word and has no further meaning to the translator.

IV. Microstatements

Each microstatement generally causes a field or group of fields in the microword to be set to a specific value. Microstatements are divided into types. Each type starts with one of a unique set of reserved words for that type. Appendix A should be referenced for a description of the values used in setting specific fields in the microinstruction.

Interrupt Control Microstatements

These single word microstatements generally set the interrupt control field in the microword, bits 23 - 20. Also, the 'Clear Interrupts from Mask Register' microinstruction (CLEARMR) uses the data bus. Table CII describes the interrupt control microstatements.

Bus Transfer Microstatements

This type of microstatement follows the syntax:

LOAD <bus destination> FROM <bus source>

There can be multiple bus transfer microstatements in a microinstruction, as long as there is only one bus source named. For this purpose, the reserved word DB as a bus source means to load the bus destination from whatever is on the data bus, while DB as a bus destination means to gate the bus source onto the data bus. Thus, the microstatement

LOAD MBR FROM IR

Table CII

Interrupt Control Microstatements[1]

Name	Meaning	Use DB	Bits 23-20 Setting
CLEARMASTR	Master clear of interrupt unit		0
CLEARALL	Clear all interrupts		1
CLEARDB	Clear all interrupts from DB data		2
CLEARMR[2]	Clear interrupts from mask register		3
CLEARVEC	Clear interrupt associated with last vector read	x	4
SETMR	Set mask register (inhibits all interrupts)		8
RESETBITMR	Bit clear mask register from DB		A
SETBITMR	Bit set mask register from DB		B
RESETMR	Clear mask register (enables all interrupts)		C
DISABLEINT	Disable interrupts		D
ENABLEINT	Enable interrupts		F

Note 1 - All these microstatements (except as noted in note 2) cause the AMD 2914 priority interrupt controller to be enabled by setting bits 15-12 to a "C".

Note 2 - CLEARMR causes bits 11-8 to be set to a "9" instead of the bits in note 1.

has the same effect as the two microstatements

LOAD DB FROM IR; LOAD MBR FROM DB

Bus transfer statements do not always use only the bus source (bits 11 - 8) and the bus destination (bits 7 - 4) fields of the microword. For example, the mask and status registers as bus source and destination are controlled by the interrupt control field. Table CIII identifies the legal bus sources and the fields that they affect, and Table CIV does the same for bus destinations.

MEMORY is a special case of bus source and destination, in that it can only be paired with MBR or MBR/DMA in a bus transfer microstatement. When used with MBR, a memory reference occurs using the MAR as the memory address. The BAR is used as the memory address if MBR/DMA is used.

Command Microstatements

The command microstatements are single word microstatements used to take certain actions or to perform predefined data transfers. These are summarized in Table CV.

Auxiliary Commands Microstatements

The auxiliary commands, along with the auxiliary functions, provide two four-bit fields in the microword which the user can decode (with appropriate hardware) to perform functions other than those inherent in the basic MIME hardware. The auxiliary commands are included to allow the user to redefine these reserved words to the translator in order to produce the

Table CIII

Bus Source Names

Name	Meaning	Microword Fields Used		
		Start Bit	Length	Value
PC	Program Counter	11	4	0
MBR[2]	Memory Buffer Register	11	4	2
MBR/DMA[2]	Memory Buffer Register	11	4	2
IR	Instruction Register	11	4	3
MEMORY	Memory read	7	4	4
ALU	Output of the ALU	11	4	6
CCR	Condition Code Register (Macro condition codes)	11	4	7
WCR	Word Count Register	11	4	B
FP	Front Panel Keyboard	11	4	C
constant	Value of the constant	11	4	D
IOBR	I/O Buffer Register	11	4	E
SR	Interrupt Status Register	23	4	6
		11	4	9
MR	Interrupt Mask Register	23	4	7
		11	4	9
UDAF0[1]	User defined auxiliary function 0	11	4	5
		19	4	0
UDAF1	User defined auxiliary function 1	11	4	5
		19	4	1
⋮				
UDAFE	User defined auxiliary function 14	11	4	5
		19	4	E
UDAFF	User defined auxiliary function 15	11	4	5
		19	4	F
DB	Whatever is currently on data bus	--	-	-

Note 1 - User defined auxiliary functions cause the bus source field in the microword, bits 11-8, to be set to '5', signifying a read from the auxiliary hardware board. The auxiliary function field, bits 19-16, determine what the auxiliary board supplies to the read command.

Note 2 - If the bus destination used with this bus source is MEMORY, then a memory read is signaled by setting bits 7-4 in the microword to '4'. MBR causes the MAR to be used as the memory address (bits 11-8 set to '1'), while MBR/DMA causes the BAR to be used as the memory address (bits 11-8 set to 'A'). The MBR is always used as the bus destination on a memory read.

Table CIV

Bus Destination Names

Name	Meaning	Microword Fields Used		
		Start Bit	Length	Value
PC	Program Counter	7	4	0
MAR[2]	Memory Address Register	7	4	1
MBR[2]	Memory Buffer Register	7	4	2
IR	Instruction Register	7	4	3
MEMORY	Memory write	7	4	5
DBR	Data Buffer Register	7	4	6
CCR	Macro Condition Code Register	7	4	7
BAR	Base Address Register	7	4	A
WCR	Word Count Register	7	4	B
FP	Front Panel display	7	4	C
IOBR	I/O Buffer Register	7	4	E
ALATCH	ALU A register latch	15	4	6
BLATCH	ALU B register latch	15	4	7
ABLATCH	ALU A & B register latches	15	4	8
DB	Gate bus source onto DB	--	-	-
SR	Interrupt Status Register	23	4	9
		11	4	9
MR	Interrupt Mask Register	23	4	E
		11	4	9
UDAF0[1]	User defined auxiliary function 0	7	4	D
		19	4	0
UDAF1	User defined auxiliary function 1	7	4	D
		19	4	1
⋮				
UDAFE	User defined auxiliary function 14	7	4	D
		19	4	E
UDAFF	User defined auxiliary function 15	7	4	D
		19	4	F
MBR/DMA[2]	Memory Buffer Register	7	4	2

Note 1 - User defined auxiliary functions cause the bus destination field in the microword, bits 7-4, to be set to 'D', signifying a write to the auxiliary hardware board. The auxiliary function field, bits 19-16, determines what the auxiliary board does with the data written to it.

Note 2 - If the bus source used with this bus destination is MEMORY, then a memory write is signaled by setting bits 7-4 in the microword to a '5'. MBR causes the MAR to be used as the memory address (bits 11-8 set to '1'), while MBR/DMA causes the BAR to be used as the memory address (bits 11-8 set to 'A'). The MBR is always used as the bus source on a memory write.

Table CV

Command Microstatements

		Microword Fields Used		
Name	Meaning	Start Bit	Length	Value
INCPC	Increment PC	15	4	0
INCMAR	Increment MAR	15	4	1
INCDMA	Increment WCR and BAR	15	4	2
TOGREAD	Toggle Read FF	15	4	3
RESETMEM	Reset Memory FF	15	4	5
DCDLOWIR	Decode lower part of IR through Mapping PROM	15	4	9
IOWRITE	Write to I/O device	15	4	D
IOREAD	Read from I/O device	15	4	E
HALT	Cease executing micro-instructions	11	4	4
ADDRESS	Put MAR onto address bus	11	4	1
ADDRESS/DMA	Put BAR onto address bus	11	4	A
INCB	Increment B latch	15	4	A
DECB	Decrement B latch	15	4	B

desired code using recognizable mnemonics. The single word auxiliary commands are summarized in Table CVI.

Set Condition Codes Microstatements

The set condition codes microstatements are described by the following syntax rules:

⟨set condition codes⟩ := ⟨condition class⟩
[: ⟨condition codes⟩]

⟨condition class⟩ := SETCC ! SETMCC

⟨condition codes⟩ := ⟨empty⟩! ⟨ccodes⟩ !
⟨condition codes⟩ , ⟨ccodes⟩

⟨ccodes⟩ := ⟨name⟩

This microstatement form is used to set either the micro condition code register (by using SETMCC), or the macro

Table CVI

Auxiliary Commands[1]

		Microword Fields Used		
Name	Meaning	Start bit	Length	Value
UDAC0	Auxiliary Command 0	23	4	0
UDAC1	Auxiliary Command 1	23	4	1
UDAC2	Auxiliary Command 2	23	4	2
UDAC3	Auxiliary Command 3	23	4	3
UDAC4	Auxiliary Command 4	23	4	4
UDAC5	Auxiliary Command 5	23	4	5
UDAC6	Auxiliary Command 6	23	4	6
UDAC7	Auxiliary Command 7	23	4	7
UDAC8	Auxiliary Command 8	23	4	8
UDAC9	Auxiliary Command 9	23	4	9
UDACA	Auxiliary Command 10	23	4	A
UDACB	Auxiliary Command 11	23	4	B
UDACC	Auxiliary Command 12	23	4	C
UDACD	Auxiliary Command 13	23	4	D
UDACE	Auxiliary Command 14	23	4	E
UDACF	Auxiliary Command 15	23	4	F

Note 1 - These commands also cause the command field of the microword, bits 15-12, to be set to 'F' to signify to select from the auxiliary command field.

condition code register (by using SETCC). It should be noted that, although the condition codes can be set using the current macro condition codes, it is not possible to set all the (micro or macro) condition codes based on the current micro condition codes. This suggests that caution should be exercised by the programmer in setting and testing micro condition codes. In general, micro condition codes should be set on one ALU operation and tested before the succeeding ALU operation, since the micro condition codes can only be preserved across the second ALU operation if SETCC is specified on the second ALU operation. The default condition code

sources are the corresponding macro condition codes, whether SETMCC or SETCC is specified. Defaults are used if:

- No condition code setting is specified in the same microinstruction as an ALU operation (in which case SETCC is understood).
- There is no colon and <condition codes> following the SETMCC or SETCC.
- The specific condition code is not mentioned in the list of ccodes following the colon, which should define what is to be the source for the setting of from one to all four of the condition codes (carry, zero, negative, and overflow).

Also, when using a shifted ALU destination, the condition codes are set based on the ALU result before it is shifted and stored. Table CVII describes the valid <ccodes> . Note that certain sources for setting the carry condition code are only meaningful if a shifted ALU destination is used (see note 3 of Table CVII).

As an example, the microstatement

```
SETMCC:WZERO,BNEG,RAM15
```

would cause the micro-zero condition code to be set based on whether the 16-bit result in the ALU is zero, the micro-negative condition code to be set based on whether the selected byte in the ALU was negative (the upper or lower byte in the ALU is disabled using the disable byte micro-statement), and the micro-carry condition code would be set to the value of bit 15 of the selected register in the ALU

Table CVII

Condition Code Names (ccodes)

Name	Meaning	Microword Fields Used		
		Start Bit	Length	Value
NEG[1]	Current macro N condition code	63	2	0
BNEG[2]	Active byte's N condition	63	2	1
NEGØ	Clear N condition code	63	2	2
NEG1	Set N condition code	63	2	3
OVER[1]	Current macro OVR condition code	61	2	0
BOVER[2]	Active byte's OVR condition	61	2	1
OVERØ	Clear OVR condition code	61	2	2
OVER1	Set OVR condition code	61	2	3
LOWZERO	Lower byte Z condition	58	4	2
UPZERO	Upper byte Z condition	58	4	3
ZERO[1]	Current macro Z condition code	58	4	4
WZERO	Word zero condition	58	4	5
ZERØ	Clear Z condition code	58	4	6
ZERO1	Set Z condition code	58	4	7
CARRY[1]	Current macro C condition code	55	4	0
CARRY*	Inverse of CARRY	55	4	1
AC1	Carry from lowest 4 bits	55	4	6
LOWCARRY	Carry from lower byte	55	4	7
UPCARRY	Carry from upper byte	55	4	8
RAMØ[3]	LSB of ALU RAM register	55	4	9
RAM15[3]	MSB of ALU RAM register	55	4	A
Q15[3]	MSB of Q register	55	4	B
RAM7[3]	Bit 7 of ALU RAM register	55	4	C
RAM8[3]	Bit 8 of ALU RAM register	55	4	D
CARRYØ	Clear C condition code	55	4	E
CARRY1	Set C condition code	55	4	F

Note 1 - If there is no source for the specific condition code, these are the default settings.

Note 2 - Bytes selected using disable byte microstatements.

Note 3 - These sources for setting the carry condition code are only meaningful when using a shifted ALU destination (see Table CXII).

Table CVIII

Disable Byte Microstatements

		Microword Field Used		
Name	Meaning	Start Bit	Length	Value
DISABLELOW	Disable lower byte	27	1	1
DISABLEUP	Disable upper byte	31	1	1

register file. The micro overflow condition code, since it is not specified, will receive the "default" value, which is the current value of the macro overflow condition code. Obviously, it is invalid to specify more than one <ccodes> for each of the four condition codes.

Disable Byte Microstatements

These single word microstatements are used to inhibit storage of the upper or lower byte of the ALU during store operations. The default is that both bytes are enabled. The microstatement options for this are defined in Table CVIII.

ALU Function Microstatements

These microstatements are used to control the operation of the ALU. These microstatements are a generalization of the functions of which the Am2901 (the ALU in MIME) is capable (Ref 1:2-6) and are necessarily limited by the capabilities of this integrated circuit. This microstatement follows the syntax

<ALU source> <ALU function> [: <carry in>] <ALU source> =
<ALU destination> [Q : <left/right Q shift in>
RAM :<left/right RAM shift in>, CENTER :

<left/right center shift in>]

The <ALU source> fields control the source operands of the ALU. Depending on the <ALU function> there may be restrictions on what pair of operands may be used or what order they may be in. The valid source operands are described in Table CIX. Valid combinations of these registers as source operands for the ALU are described in the matrix of Figure C1.

ALU functions are described in Table CX. Note that the arithmetic add and subtract ALU functions require the programmer to specify a value for the carry in value. Table CXI defines permissible names for the carry in values.

ALU destinations are described in Table CXII. Note that if the value stored at Q and/or the selected register is the shifted output of Q and/or the register, the shift linkages must be specified. Tables CXIII through CXVIII describe these linkages. Also remember that condition codes set during this microinstruction are based on the ALU result before it is shifted and stored.

Sequence Microstatements

Sequence microstatements are governed by the hardware used in MIME to determine the next microaddress - the Am2909/2911 microprogram sequencers and the Am29811 16-way branch controller (Ref 1:2-74,2-261). Sequence microstatements are divided into three types - the multi-way branch, the unconditional microstatement, and the conditional microstatement. These can be described by the following syntax:

Table CIX

ALU Source Operands

Name	Meaning
A	Register addressed by the A latch
B	Register addressed by the B latch
Q	Q register in the ALU
D or DBR	Data buffer register
ø	Integer zero

ALU Source 2

ALU Source 1	A	B	Q	DBR	ø
A	0	1	1	3	3
B	3	0	0	0	3
Q	3	0	0	3	3
DBR	1	0	1	0	1
ø	1	1	1	3	0

Matrix Value	Meaning
0	Invalid operand combination in all cases
1	Acceptable operands for "-", "+", NRAS, OR, AND, XOR, and XNOR ALU functions
3	Acceptable operands for "-", "+", OR, AND, XOR, and XNOR ALU functions

Figure C1.

Valid ALU Source Combinations

Table CX

ALU Functions

Name	Meaning	Microword Fields Used		
		Start Bit	Length	Value
+	$R + S + C_{in}$ [3]	30	3	0
-	See notes 2 and 3	30	3	2
OR	R V S	30	3	3
AND	R $\wedge$ S	30	3	4
NRAS	R* $\wedge$ S	30	3	5
XOR	R $\forall$ S	30	3	6
XNOR	(R $\forall$ S)*	30	3	7

Note 1 - R represents ALU source 1.
S represents ALU source 2.
C_{in} represents the value of the carry in.

Note 2 - If the value in the matrix of Figure C1 for the ALU source operand pair is a 1, then the field value is 1, and the ALU function is $S + R^* + C_{in}$ (or the arithmetic operation S-R). If the matrix value is 3, then the field value is 2, and the ALU function is $R + S^* + C_{in}$ (or the arithmetic operation R-S).

Note 3 - Requires that the carry in be specified for this ALU operation.

Table CXI

ALU Carry In
(Required on ALU "+" and "-" functions)

Name	Meaning	Microword Fields Used		
		Start Bit	Length	Value
MCARRY	Micro carry condition code	50	3	2
MCARRY*	Inverse of micro carry condition code	50	3	3
CARRY	Macro carry condition code	50	3	4
CARRY*	Inverse of macro carry condition code	50	3	5
$\emptyset$ [1]	A constant low	50	3	6
1	A constant high	50	3	7

Note 1 - FALSE is a synonym for $\emptyset$.

Table CXII

ALU Destinations

Name[1]	Meaning	Microword Fields Used		
		Start bit	Length	Value
QF	Result stored in Q, result available as ALU output	34	3	0
ØF	No storage of result, result available as ALU output	34	3	1
BA	Result stored in B, A register available as ALU output	34	3	2
BF	Result stored in B, result available as ALU output	34	3	3
RSQBF[4,2]	Store Q right shifted back into Q, store result right shifted into B, result available as ALU output	34	3	4
RSBF[2]	Store result right shifted back into B, result available as ALU output	34	3	5
LSQBF[5,3]	Store Q left shifted back into Q, store result left shifted back into B, result available as ALU output	34	3	6
LSBF[3]	Store result left shifted back into B, result available as ALU output	34	3	7

Note 1 - The name is formed based on the following pattern:

{LS=left shift output
RS=right shift output
blank=output not shifted} {Storage Destination
Q=Q register
Ø=no storage
B=B register
QB=both Q and B registers} {ALU output available to data bus
F=result of ALU function
A=contents of A register}

Note 2 - Requires RAM right shift linkage and CENTER right shift linkage.

Note 3 - Requires RAM left shift linkage and CENTER left shift linkage.

Note 4 - Requires Q right shift linkage.

Note 5 - Requires Q left shift linkage.

Table CXIII

Q Register Left Shift In

Name	Meaning	Microword Fields Used		
		Start bit	Length	Value
RAM15	Bit 15 of the register addressed by the B latch	46	3	0
Q15	Bit 15 of Q register	46	3	1
CARRY	Macro C condition code	46	3	2
Ø	A constant low	46	3	3
1	A constant high	46	3	4

Table CXIV

Q Register Right Shift In

Name	Meaning	Microword Fields Used		
		Start bit	Length	Value
RAMØ	Bit 0 of the register addressed by the B latch	46	3	0
QØ	Bit 0 of Q register	46	3	1
CARRY	Macro C condition code	46	3	2
Ø	A constant low	46	3	3
1	A constant high	46	3	4
DBR15	Bit 15 of the data buffer register	46	3	6
RAM8	Bit 8 of the register addressed by the B latch	46	3	7

Table CXV

RAM Left Shift In

Name	Meaning	Microword Fields Used		
		Start bit	Length	Value
RAM15	Bit 15 of the register addressed by the B latch	42	3	0
Q15	Bit 15 of Q register	42	3	1
CARRY	Macro C condition code	42	3	2
Ø	A constant low	42	3	3
1	A constant high	42	3	4
Q7	Bit 7 of Q register	42	3	5
RAM7	Bit 7 of the register addressed by the B latch	42	3	7

Table CXVI

RAM Right Shift In

Name	Meaning	Microword Fields Used		
		Start bit	Length	Value
RAMØ	Bit 0 of the register addressed by the B latch	42	3	0
QØ	Bit 0 of Q register	42	3	1
CARRY	Macro C condition code	42	3	2
Ø	A constant low	42	3	3
1	A constant high	42	3	4
DBR15	Bit 15 of the data buffer register	42	3	6
RAM8	Bit 8 of the register addressed by the B latch	42	3	7

Table CXVII

Center Left Shift In

Name	Meaning	Microword Fields Used		
		Start bit	Length	Value
Ø	A constant low	37	2	0
CARRY	Macro C condition code	37	2	1
RAM7	Bit 7 of the register addressed by the B latch	37	2	3

Table CXVIII

Center Right Shift In

Name	Meaning	Microword Fields Used		
		Start bit	Length	Value
Ø	A constant low	37	2	0
CARRY	Macro C condition code	37	2	1
DBR7	Bit 7 of the data buffer register	37	2	2
RAM8	Bit 8 of the register addressed by the B latch	37	2	3

<multi-way branch> := ON<branch control> GO TO <label>

<conditional statement> := IF <condition> THEN

<unconditional statement> [ELSE

<unconditional statement>]

<unconditional statement> := GO TO <label> !

GO TO <label> : POP! GO TO REGISTER !

GO TO START ! GO TO ZERO ! GO TO MAP !

CALL <label> ! CALL REGISTER ! RETURN ! POP !

CONTINUE! PUSH ! LOADCNTR FROM<bus source> !

PUSHLDCNTR FROM <bus source> !

LOOP AT <label> ! LOOP AT FILE !

LOOP AT FILE : DECREMENT

Due to the capabilities of the Am2909/2911 microprogram sequencers (Ref 1:2-74) used in the MIME, not all combinations of unconditional statements are compatible with a conditional microstatement.

The multi-way branch can be a 2-, 4-, 8-, or 16-way branch into a branch table that starts at the label specified. The label, when it is defined, must be declared to be a multi-way branch label of the required or higher type, as specified by the branch control. The branch control specifies which bits are to be OR'ed with the lower one, two, three, or four bits of the label address in order to derive the destination of the branch. The branch control consists of one to four bit names from one of the following groups (multiple bit names are separated by a comma):

MSB			LSB	
IR3	IR2	IR1	IR0	(Instruction register)
IR6	IR5	IR4	IR3	
IR9	IR8	IR7	IR6	
IRC	IRB	IRA	IR9	
IRF	IRE	IRD	IRC	
IV3	IV2	IV1	IV0	(Interrupt vector)
Z	N	C	V	(Macro condition codes)

As an example, if the branch control was

IR3, IR1

then bits three and one from the instruction register would be OR'ed with the two least significant bits of the label microaddress to derive the destination. Since the minimum value to be OR'ed in from these bits would be a '0' and the maximum a '3', this would require that the label microaddress be declared as a minimum of a 4-way branch in order that the label would be assigned an address with the lower two bits as zero.

There are 17 unconditional transfer microstatements, which are described below:

- CALL <label> - The next sequential microinstruction address is pushed onto the sequencer stack and control transfers to the microinstruction at the microaddress specified by <label> .
- CALL REGISTER - The next sequential microinstruction address is pushed onto the sequencer stack and control transfers to the microinstruction at the microaddress contained in the sequencer register. This register always contains the value from the microaddress field of the previous microinstruction unless the previous

microinstruction was GO TO MAP or GO TO START.

- GO TO <label> - Control is transferred to the microinstruction at the microaddress specified by <label>.
- GO TO <label> : POP - Same as the previous microstatement, except that the sequencer stack is POP'ed. This is used as an alternate method of exiting a loop and maintaining the stack when a program loops using the LOOP AT FILE microstatement.
- GO TO REGISTER - Control is transferred to the microinstruction at the microaddress contained in the sequencer register.
- GO TO START - Control is transferred to the microinstruction at the microaddress specified by the start up address switches in the MIME. This is also the microaddress that the machine is set to by a front panel RESET.
- GO TO ZERO - Control is transferred to the microinstruction at microaddress 000.
- GO TO MAP - Control is transferred to the microinstruction at the microaddress specified by the output of the mapping PROM.
- RETURN - Control is transferred to the microinstruction at the microaddress that is on the top of the sequencer stack. The stack is POP'ed.
- CONTINUE - The microprogram counter is incremented by one and control transfers to the microinstruction at that microaddress.

- POP - The value on the top of the sequencer stack is discarded and control transfers to the next sequential microinstruction.
- LOADCNTR FROM <bus source> - Load the Micro Loop Counter with the eight least significant bits of the <bus source> and transfer control to the next sequential microinstruction.
- PUSH - The next sequential microinstruction's micro-address is PUSH'ed onto the sequencer stack and control transfers to the next sequential microinstruction. This is used to set up a loop that will operate using the LOOP AT FILE instruction.
- PUSHLDCNTR FROM <bus source> - Same as the PUSH micro-statement, except that the least significant eight bits of the <bus source> value is loaded into the Micro Loop Counter.
- LOOP AT <label> - The Micro Loop Counter is decremented and control transfers to the microinstruction at the microaddress specified by <label> .
- LOOP AT FILE - Control transfers to the microinstruction at the microaddress that is currently on the top of the sequencer stack.
- LOOP AT FILE : DECREMENT - Same as LOOP AT FILE, except that the Micro Loop Counter is also decremented.

The unconditional microstatements GO TO ZERO, GO TO MAP, LOADCNTR FROM <bus source> , CONTINUE, and GO TO <label> are true unconditional microstatements. The remaining ones are

generated by the translator by using either a constant true or a constant false as the condition on a conditional microstatement. Error messages from the translator may mention conditions or conditional statements when, at first glance, the programmer may think that only unconditional microstatements are used.

In a conditional microstatement, the conditional test can be based on the condition codes, I/O status, or bits in various registers. Tables CXIX, CXX, and CXXI describe the valid conditions. Each of these may be preceeded by the name NOT, in which case the polarity field of the microword, bit 60, is inverted.

Because of the capabilities of the Am2909/2911 microprogram sequencers and the Am29811 next address control unit used in the MIME, not all of the unconditional microstatements are valid as part of an IF-THEN or IF-THEN-ELSE construct.

The following are valid unconditional transfer statements that can be used in the IF-THEN construct, along with typical uses of them:

- CALL <label> - This is used as a conditional jump to subroutine.
- GO TO <label> - This is used for conditional branching.
- GO TO START - This is used to conditionally transfer to the start up microaddress set in the switches of MIME.
- LOOP AT <label> - Using CT* as the condition, this instruction will go to<label> and decrement the Micro

Table CXIX

I/O Conditions[1]

		Microword Fields Used		
Name	Meaning	Start Bit	Length	Value
DMAOVER	BAR incremented beyond max memory address	54	3	0
DMATERM	WCR incremented to zero	54	3	1
DMAREAD	State of the READ FF	54	3	2
TTYSTATUS	TTY ready state from serial port	54	3	3
IRQ*	Interrupt request	54	3	7

Note 1 - These all cause bits 59-56 to be set to 2 in order to select I/O conditions (I/O TC MUX).

Table CXX

ALU Conditions[1]

		Microword Fields Used		
Name	Meaning	Start Bit	Length	Value
MFØ	Micro FØ latch	55	4	1
MALTB	Micro A less than B latch	55	4	2
MAGTB	Micro A grtr than B latch	55	4	3
MCARRY	Micro Carry condition code latch	55	4	4
MOVER	Micro Over condition code latch	55	4	5
MZERO	Micro Zero condition code latch	55	4	6
MNEG	Micro Negative condition code latch	55	4	7
FØ	Macro FØ latch	55	4	9
ALTB	Macro A less than B latch	55	4	A
AGTB	Macro A grtr than B latch	55	4	B
CARRY	Macro Carry condition code latch	55	4	C
OVER	Macro Over condition code latch	55	4	D
ZERO	Macro Zero condition code latch	55	4	E
NEG	Macro Negative condition code latch	55	4	F

Note 1 - These all cause bits 59-56 of the microword to be set to 3 to select ALU conditions (ALU TC MUX).

Table CXXI

Miscellaneous Conditions

		Microword Fields Used		
Name	Meaning	Start Bit	Length	Value
BP1	Frontpanel breakpoint 1 switch on	59	4	0
BP2	Frontpanel breakpoint 2 switch on	59	4	1
MARØ	MAR bit zero	59	4	4
MEMORY	Memory FF set	59	4	5
CT*	Micro Loop Counter is currently zero	59	4	6
IR4	Instruction register bit 4	59	4	7
IR5	Instruction register bit 5	59	4	8
IR7	Instruction register bit 7	59	4	9
IR15	Instruction register bit 15	59	4	A
TRUE	A constant high	59	4	F
1	Same as TRUE	59	4	F

Loop Counter until CT* is true. This signifies that the counter is zero, and the next sequential microinstruction is executed. The Micro Loop Counter would be initially set using the LOADCNTR microinstruction. Note that the loop counter is tested at the start of execution of this microinstruction, while the decrementing occurs at the end of the microinstruction.

- RETURN - This is used to conditionally return from a subroutine.
- GO TO <label> : POP - This is used for terminating a loop that was set up using a PUSH microinstruction and executed using a LOOP AT FILE microinstruction. Using this instruction to exit the loop maintains the sequencer

stack by POP'ing the loop address off the top.

The following are valid unconditional statement pairs that can be used in the IF-THEN-ELSE construct, along with typical uses of them:

- PUSH/PUSHLDCNTR FROM <bus source> - This is used to set up a loop to be executed using the LOOP AT FILE microinstruction. Conditionally, the Micro Loop Counter can be loaded to execute the loop a specific number of times, and the LOOP AT FILE: DECREMENT microinstruction used to execute and terminate the loop.
- CALL REGISTER/CALL <label> - This allows the selection of one of two subroutines to execute.
- GO TO REGISTER/GO TO <label> - This allows the selection of one of two transfer microaddresses.
- LOOP AT FILE: DECREMENT/POP - This would be used to terminate a loop that was set up using the PUSHLDCNTR microinstruction. Using CT* as the condition, the Micro Loop Counter would be decremented and control transferred to the microaddress on the top of the sequencer stack as long as the counter is non-zero. When the Micro Loop Counter reaches zero, the loop address is POP'ed off the top of the sequencer stack and the next sequential microinstruction is executed.
- LOOP AT FILE/POP - This operates similar to the previous microinstruction pair, except that some condition other than CT* would be used to terminate the loop that was set up using a PUSH microinstruction. The Micro Loop Counter is not used in this case.

V. Pseudo-Operations

Pseudo-operations are directives to the translator to control the translation process. These must be on separate input cards from any other source statements and are not terminated by a period as are other source microinstructions.

ORG <constant>

The ORG pseudo-operation causes the Micro Program Counter to be set to the value of the constant.

PAGE <title>

The PAGE pseudo-operation causes a page eject to be generated and a new page header printed on the output listing. Optionally, a title of up to 50 characters may be included on the card, which is printed on the second line of this new page heading and each succeeding one until a new title is specified.

PASS3

The PASS3 pseudo-operation causes post-processing of the translator output to occur. This includes formatting the resulting object code for programming PROM's, detecting unused imbedded blocks of control store memory, and generating the data necessary to program the mapping PROM's for MIME. PASS3 is described fully in Chapter VI of this appendix.

VI. Using the Translator

This chapter describes how to use the translator. Discussions of executing the translator, error processing, and post-processing are included. The formats and content of the input and output files are also described.

Executing the Translator

The MIME translator is a program written in Pascal and designed to run of the CDC Cyber Series Computer System under the NOS/BE operating system (Ref 2). The program requires a 65000_8 field length to compile and a variable field length to execute, depending upon the size of the MIME language source program. The discussion below assumes that the user has access to a compiled binary version of the program, and that the Pascal Library (PASCLIB, ID=AFIT) has been attached and declared as a LIBRARY.

Assuming the translator program file name is MIMEBIN the following calling sequence will execute the translator program:

MIMEBIN, INP, OUT, OBJ, OPC.

where

INP = MIME language source file name, default = INPUT.

OUT = Output listing file name, default = OUTPUT.

OBJ = Output object file name, default = OBJECT.

OPC = Input file containing mnemonics from which to

create mapping PROM, default = OPCODES.

The formats of these files will be described later.

If the execution error, "Runtime Stack Overflow", is received, this indicates that the field length allowed for the program is too small. Extending the field length using the RFL command (EFL under Intercom) and restarting the program should cure this problem.

Error Processing

In general, errors are of two types:

- Invalid name or operation in a specific field
- Multiple use of a single field in the microword. As an example, it is not possible to use a constant as a bus source in the same microinstruction as a conditional transfer microstatement, since they both use the same fields in the microword (constant = bits 63 - 48, condition = bits 63 - 52).

When an error is encountered in the translation of a microinstruction, a message is printed on the output listing before the printing of the source microinstruction. A listing of error messages and a further explanation of their meaning (if needed) is contained in Attachment C-C. Additionally, a "V" character and the error number are printed on the line following the error message. The "V" points to the name or delimiter in which the error was detected.

Some errors cause the translator to enter into a "scan" mode. In these cases (see Attachment C-C for a listing of errors which cause this), the translator is not capable of

determining exactly what the intentions of the programmer were. In order to attempt to recover from the error, the translator scans ahead until it encounters an end-of-micro-statement (a semicolon) before resuming the translation. In all cases, once an error is detected, the programmer should be aware that it may cause the translator to find non-existent errors or to miss existing errors in remainder of the micro-statement. However, recovery should be complete by the end of the bad microstatement.

Post-processing

An additional option is available to perform a "third" pass on the translator output and produce an OBJECT file of microcode for programming PROM's or for directly loading control store. This option performs the following functions:

- Lists all the holes in the control store memory. With the microcode in microaddress order, any noncontiguous sections would indicate a hole where there was no microcode generated. This could be caused by the use of the ORG pseudo-operation or by the automatic handling of the microprogram counter by the translator to accomodate the multi-way branch labels. Since control store is a limited resource in MIME, the programmer may want to reorganize the microinstructions in order to minimize these holes.
- Inserts No Operation (NOP) instructions into the holes. This is meant to aid a PROM programmer. Since a PROM programmer does not operate using any address data in

the file, but instead fills the PROM continuously from the input data, any holes in the microcode must be filled with NOP's to present a continuous stream of microwords for the PROM.

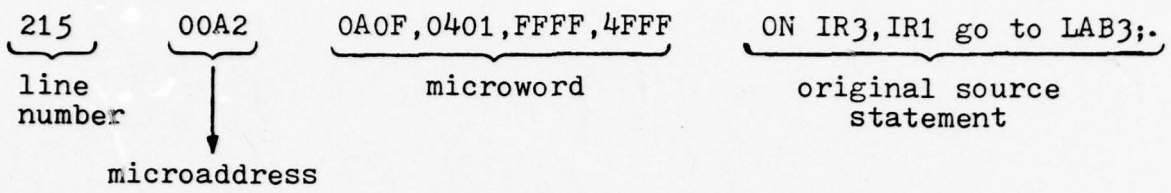
- Create the mapping PROM data. File OPCODES should contain the mnemonics for the instructions in ascending order. In the source code for the translation, the microcode for emulation of an opcode should start at a label formed by concatenating a per cent sign ("%") and the opcode mnemonic. The PASS3 processing attempts to match the mnemonics from the OPCODES file with corresponding labels in the translated program in order to map the opcode to a microaddress where the microcode to emulate the opcode is located. If the label is not found in the program, then PASS3 attempts to use the address for the label "%NOP", if it is defined in the program. Appropriate messages are written to the file OUTPUT.

File Formats

INPUT File. This file is composed of 80 column card images. The microinstruction source statements on this file are free format, as long as they conform to the structure standards discussed in Chapter II of this appendix.

OUTPUT File. This file consists of the original source microinstructions, the corresponding microword generated, a line number, and any error messages for each source microinstruction. A typical line on this file is formatted as

follows:



Additionally, a symbol table, a list of redefinitions, the results of the post-processing, an index of page titles, and a list of redefined reserved words are printed.

The symbol table is formatted as follows:

SYMBOL TABLE

SYMBOL	ADDRESS	TYPE
LAB1	00A0	16
LAB2	0100	20
LAB3	0093	0

label name — hexadecimal microaddress — label type

Label types are as follows:

0 - No special type
2 - Destination of 2-way branch
4 - Destination of 4-way branch
8 - Destination of 8-way branch
16 - Destination of 16-way branch
20 - Loop
21 - Externally declared

The redefinition table is formatted as follows:

USER REDEFINITIONS

ORIGINAL RESERVED WORD	REDEFINITION
MAR	MEMADDR
UDAFC	MIME3

reserved word from Attachment A — new mnemonic for this reserved word

If the PASS3 pseudo operation is not used to invoke the post-processing of the translation, the message

PASS3 NOT INVOKED - <OBJECT> FILE IS EMPTY

is printed. If PASS3 is called, the following sequence of messages would be a typical output:

```
PASS3 BEGUN
<OPCODES>  CONTAINED 256 LABELS
PASS3 COMPLETE - FREE MEMORY BLOCKS FOLLOW
ADDR       SIZE
0482         14
04C7          1
04CA          6
051C          4
FORMATTED MICROCODE IS ON FILE  <OBJECT>
```

The second line of the above message indicates what the post-processing found on the file <OPCODES> . This file is used along with the results of the translation to attempt to construct the data for a mapping PROM. See the Post-processing section of this chapter.

The description of the free memory blocks on lines 5 - 8 above shows microaddresses where no microcode was generated. This may be either because the ORG pseudo-operation was used more than once or because the translator encountered one or more multi-way branch labels. The post-processing inserts a "do nothing" microinstruction in these locations, but the programmer may want to reorganize the microinstructions to minimize these unused memory blocks.

The resulting microcode formatted for programming PROM's is on the file OBJECT, as signified by the last line of the message.

If the microprogrammer has used the PAGE pseudo-operation with its optional titles, an index of the page numbers where new titles were introduced is printed. An example of this index is

```
INDEX

PAGE        TITLE

  1
  1         EXPLANATORY COMMENTS
  4         REDEFINITION OF AUXILIARY FUNCTIONS
  6         INITIALIZATION
  .            .
  .            .
  .            .
```

The first line is always page 1 with a blank title.

OBJECT File. This file is only filled if the PASS3 pseudo-operation is used to post-process the translation results. If PASS3 is invoked, this file contains the MIME microcode. The format of this file is compatible with the PROM data manipulation program of Appendix N.

First on the file is a message similar to

```
;MIME ROM MACHINE CODE    09/12/79   18:12:20
```

signifying the start of the microcode, the date and the time. This is followed by the microcode in the format

```
↓0123,4557,89AB,CDEF↓;1234↓--5678
```

64 bit microword contents, in hexadecimal characters	micro-address of this microword	address of this microword for programming 1K PROM's

Following the microcode comes the message

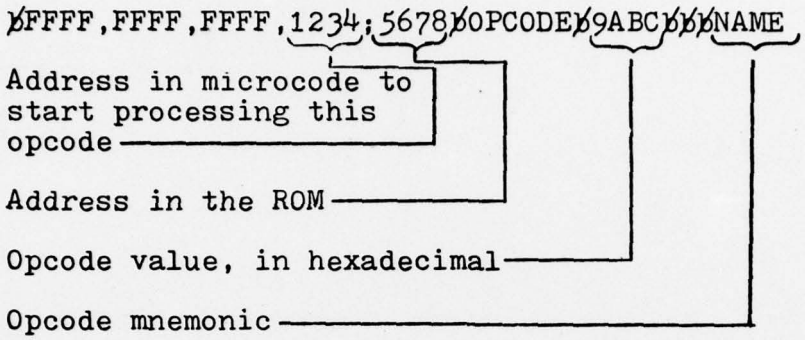
```
;MIME MAPPING PROM CODE     09/12/79     18:12:20
```

AD-A080 420 AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OH SCHOO--ETC F/G 9/2
MIME: MICROPROGRAMMABLE MINICOMPUTER EMULATOR. PHASE II. VOLUME--ETC(U)
DEC 79 T R HOYT, D A MYERS
UNCLASSIFIED AFIT/GCS/EE/79-11-VOL-2 NL

3 OF 3
AD
A080420

END
DATE
FILMED
3 - 80
DDC

signifying the start of data for programming the mapping PROM's, the date and the time. Next comes the microcode for programming the mapping PROM's in the format



The final message on the file will be similar to

; <OPCODES> CONTAINED 256 LABELS

showing how many mnemonics were found on the file OPCODES.

OPCODES File. This file contains the mnemonics used to create the mapping PROM data in free format separated from one another by one or more spaces. These must be in ascending order by numeric opcode (i.e., the first mnemonic on this file must correspond to opcode 00, the second to opcode 01, etc.). The PASS3 processing precedes each mnemonic by a per cent sign ("%") and looks for the resulting label in the translation results. The address of that label is used to create the mapping PROM data in the OBJECT file.

References

1. The Am2900 Family Data Book with Related Support Circuits. Sunnyvale, CA: Advanced Micro Devices, Inc., 1978.

2. NOS/BE Version 1 Reference Manual. Number 60493800, Revision F. St. Paul, MN: Control Data Corp., 1978.

Attachment C-A

List of Reserved Words

This attachment lists words that have a special meaning to the translator. Use of these words as labels must be avoided. If the word has an entry in the column marked "not redefinable", that word can not be redefined in the redefinition section. All other words can be redefined.

Reserved Word	Not Redefinable	Description and/or use
A		ALU operand; the register addressed by the A address latch.
ABLATCH		Bus destination for setting the A and B address latches.
AC1		Ccode for setting carry condition code from the auxiliary carry (carry out of lower 4 bits).
ADD		ALU function.
ADDRESS		Command for gating the MAR onto the address bus.
ADDRESS/DMA		Command for gating the BAR onto the address bus.
AGTB		Macro condition of the contents of the register addressed by the A address latch greater than the contents of the register addressed by the B address latch.
ALATCH		Bus destination for setting the A address latch.
ALTB		Macro condition of the contents of the register addressed by the A address latch less than the contents of the register addressed by the B address latch.
ALU		Bus source for gating the ALU output onto the data bus.
AND		ALU function.
AT		Keyword used as part of a sequence microstatement such as "LOOP AT--".
B		ALU operand; the register addressed by the B address latch.
BA		ALU destination; result goes to the B register and the A register is available as the ALU output.
BAR		Bus destination for loading the base address register.

BF	ALU destination; result goes to the register addressed by the B latch and is available as the ALU output.
BLATCH	Bus destination for setting the B address latch.
BNEG	Ccode for setting the negative condition code from the negative status of the enabled byte.
BOVER	Ccode for setting the overflow condition code from the overflow status of the enabled byte.
BP1	Condition of the front panel breakpoint 1 switch.
BP2	Condition of the front panel breakpoint 2 switch.
C	Branch control for multi-way branch, based on status of macro carry condition code.
CALL	Keyword for sequence microstatement to transfer to a subroutine.
CARRY	Macro carry condition code; used *for:* (1) Ccode to set carry condition codes. (2) Condition. (3) Carry-in to an add or subtract ALU operation. (4) Shift in for a shifted ALU destination; can be used in the Q, RAM, or CENTER shift linkage.
CARRYØ	Ccode to clear carry condition code.
CARRY1	Ccode to set carry condition code.
CARRY*	The inverted status of the macro carry condition code; used for: (1) Ccode for setting carry condition code. (2) Carry-in to an add or subtract ALU operation.
CCR	Bus source and bus destination

		using the macro condition code register.
CENTER		Keyword to identify CENTER shift linkage on a shifted ALU destination.
CLEARALL		Interrupt control microstatement to clear all interrupts.
CLEARDB		Interrupt control microstatement to clear interrupts based on the data bus information.
CLEARMASTR		Interrupt control microstatement to clear interrupts based on mask register data.
CLEARVEC		Interrupt control microstatement to clear the interrupt associated with the last interrupt vector read.
CONTINUE		Sequence microstatement.
CT*		Condition of micro loop counter; true when counter is not zero.
D		Data buffer register; used as: (1) ALU operand. (2) Bus destination.
DB		Data bus; used as: (1) Bus source. (2) Bus destination.
DBR		Same as D.
DBR15		Most significant bit (sign bit) of data buffer register; used for right shift linkage for Q and RAM on a right shifted ALU destination.
DBR7		Center shift linkage for a right shifted ALU destination.
DCDLOWIR		Command to use low order half of IR (instead of high order half) as address into mapping PROM.
DEBUG	X	Special maintenance programmer pseudo-operation.
DECB		Command to decrement the B

		address latch by one.
DECREMENT		Keyword in a sequence microstatement, as in LOOP AT FILE: DECREMENT.
DISABLEINT		Interrupt control microstatement to disable interrupts.
DISABLELOW		Disable byte microstatement to disable least significant byte.
DISABLEUP		Disable byte microstatement to disable most significant byte.
DMAOVER		Condition signifying the BAR overflowed.
DMAREAD		Condition of DMA READ FF.
DMATERM		Condition signifying WCR has counted to zero.
DUMMYCONST		Used internally in the translator to represent constants.
ELSE		Keyword in a conditional sequence microstatement.
ENABLEINT		Interrupt control microstatement to enable interrupts.
END	X	Last card of a microprogram.
FALSE		Carry-in of integer zero to an ALU add or subtract operation.
FILE		Keyword in a sequence microstatement, as in LOOP AT FILE.
FP		Front panel; used for: (1) Bus source (using the front panel keyboard). (2) Bus destination (using the front panel macro displays).
FROM		Keyword in a bus transfer microstatement used to identify the bus source.
FØ		Macro condition of the least significant bit of the ALU output.

GO	Keyword used in sequence microstatement, such as GO TO REGISTER.
HALT	Command to stop execution of microinstructions.
IF	Keyword used in a conditional sequence microstatement.
INCB	Command to increment the B address latch by one.
INCDMA	Command to increment both the BAR and WCR by one.
INCMAR	Command to increment the MAR by one.
INCPC	Command to increment the PC by one.
IOBR	I/O buffer register, used as: (1) bus source. (2) bus destination.
IOREAD	Command to read from an I/O device.
IOWRITE	Command to write to an I/O device.
IR	Instruction register, used as: (1) bus source. (2) bus destination.
IRQ*	Condition of interrupt request signal; true when an interrupt occurs.
IRØ	Branch control for multi-way branch, using bit Ø (least significant) of the IR.
IR1	Branch control for multi-way branch, using bit 1 of IR.
IR2	Branch control for multi-way branch, using bit 2 of IR.
IR3	Branch control for multi-way branch, using bit 3 of IR.
IR4	Bit 4 of IR, used for: (1) Branch control for multi-way branch.

	(2) Condition.
IR5	Bit 5 of IR, used for (1) Branch control for multi-way branch. (2) Condition.
IR6	Branch control for multi-way branch, using bit 6 of IR.
IR7	Bit 7 of IR (most significant bit of least significant byte), used for: (1) Branch control for multi-way branch. (2) Condition.
IR8	Branch control for multi-way branch, using bit 8 of IR.
IR9	Branch control for multi-way branch, using bit 9 of IR.
IR10	Branch control for multi-way branch, using bit 10 of IR.
IR11	Branch control for multi-way branch, using bit 11 of IR.
IR12	Branch control for multi-way branch, using bit 12 of IR.
IR13	Branch control for multi-way branch, using bit 13 of IR.
IR14	Branch control for multi-way branch, using bit 14 of IR.
IR15	Bit 15 (most significant bit) of IR, used for: (1) Branch control for multi-way branch. (2) Condition.
IVØ	Branch control for multi-way branch, using bit Ø (least significant) of interrupt vector.
IV1	Branch control for multi-way branch, using bit 1 of interrupt vector.
IV2	Branch control for multi-way branch using bit 2 of interrupt

	vector.
IV3	Branch control for multi-way branch, using bit 3 of interrupt vector.
LOAD	Keyword to start a bus transfer microstatement.
LOADCNTR	Keyword of a sequence micro-statement to load the micro loop counter.
LOOP	(1) Keyword for a sequence micro-statement, such as LOOP AT FILE. (2) Label type, identifying the attached label as a destination for a LOOP AT FILE micro-statement and causing the microaddress of the next microinstruction to be pushed on the stack.
LOWCARRY	Ccode for setting the carry condition code from the carry out of the least significant byte of the ALU result.
LOWZERO	Ccode for setting the zero condition code based on whether the least significant byte of the ALU result is zero.
LSBF	Shifted ALU destination, causing the B register to be left shifted by one bit and the result of the ALU operation, after being shifted left by one bit, to be available as the ALU output.
LSQBF	Shifted ALU destination, causing the B register to be left shifted by one bit and the result of the ALU operation, after being shifted left by one bit, to be stored in the Q register and be available at the ALU output.
MAGTB	Micro condition code of the register addressed by the A latch being greater than the value in the register addressed by the B latch.

MALTB	Micro condition code of the value in the register addressed by the A latch being less than the value in the register addressed by the B latch.
MAP	Keyword in a sequence micro-statement to indicate that the branch microaddress is the output of the mapping PROM (e.g., GO TO MAP).
MAR	Bus destination for loading the memory address register.
MARØ	Condition based on the state of bit Ø (least significant) of the memory address register.
MBR	Memory buffer register, used as: (1) Bus source. (2) Bus destination. When used as the source or destination of a memory transfer, the MAR is used on the address bus as the memory address.
MBR/DMA	Memory buffer register, used as: (1) Bus source. (2) Bus destination. When used as the source or destination of a memory transfer, the BAR is used on the address bus as the memory address.
MCARRY	Micro carry condition code, used as: (1) Condition. (2) Carry in to an add or subtract ALU operation.
MCARRY*	Carry in to an add or subtract ALU operation, using the inverted value of the micro carry condition code.
MEMORY	(1) Bus destination. (2) Bus source. (3) Condition of the memory read FF.
MFØ	Micro condition of the least significant bit of the ALU output.

MNEG		Condition of the micro negative condition code.
MOVER		Condition of the micro overflow condition code.
MR		Mask register for interrupts, used for: (1) Bus source. (2) Bus destination.
MZERO		Condition of the micro zero condition code.
N		Branch control for multi-way branch, based on the status of the macro negative condition code.
NEG		Macro negative condition code, used for: (1) Condition. (2) Ccode for setting the negative condition code based on the state of the macro negative condition code.
NEGØ		Ccode to clear the negative condition code.
NEG1		Ccode to set the negative condition code.
NOT		Keyword used preceeding a condition to negate the condition.
NRAS		ALU function.
ON		Keyword used to begin a multi-way branch microstatement.
ORG	X	Pseudo-operation to set the micro program counter during the translation process.
OR		ALU function.
OVER		Macro overflow condition code, used for: (1) Condition. (2) Ccode to set the overflow condition code based on the status of the macro overflow condition code.

OVERØ		Ccode to clear an overflow condition code.
OVER1		Ccode to set an overflow condition code.
PAGE	X	Pseudo-operation to cause a page eject on the output listing.
PASS3	X	Pseudo-operation to invoke post processing of the object file.
PC		Program counter, used as: (1) Bus source. (2) Bus destination.
POP		Keyword used in sequence micro-statements to pop the micro stack.
PROGRAM	X	Keyword to identify the end of the redefinition section and the start of the program section.
PUSH		Keyword in a sequence micro-statement to push the next micro-address on the micro stack.
PUSHLDCNTR		Keyword of a sequence micro-statement to load the micro loop counter and push the next micro-address on the micro stack.
QF		ALU destination, where the result of the ALU operation is stored in the Q register and is also available as the ALU output.
QØ		Bit Ø (least significant) of the Q register, used with a right shifted ALU destination as the (1) Q right shift linkage. (2) RAM right shift linkage.
Q7		Bit 7 of the Q register, used as RAM left shift linkage in a left shifted ALU destination.
Q15		Bit 15 (most significant) of the Q register; valid only when using a left shifted ALU destination in the same microinstruction; used for: (1) Ccode for setting a carry condition code based on the

	state of bit 15 of the Q register. (2) Q left shift linkage. (3) RAM left shift linkage.
Q	Q register in the ALU, used for: (1) ALU operand. (2) Keyword to identify the Q shift linkage of a shifted ALU destination.
RAM	Keyword used with a shifted ALU destination to identify the RAM shift linkage.
RAMØ	Bit Ø (least significant) of the ALU output; valid only when using a right shifted ALU destination in the same microinstruction; used for: (1) Ccode for setting a carry condition code based on bit Ø of the ALU output. (2) Q right shift linkage. (3) RAM right shift linkage.
RAM7	Bit 7 of the ALU output; valid only when using a left shifted ALU destination in the same microinstruction; used for: (1) Ccode to set a carry condition code based on bit 7 of the ALU output. (2) Q left shift linkage. (3) RAM left shift linkage.
RAM8	Bit 8 of the ALU output; valid only when using a right shifted ALU destination in the same microinstruction; used for: (1) Ccode for setting a carry condition code based on bit 8 of the ALU output. (2) Q right shift linkage. (3) RAM right shift linkage. (4) CENTER right shift linkage.
RAM15	Bit 15 (most significant) of the ALU output; valid only when using a left shifted ALU destination; used for: (1) Ccode for setting a carry condition code based on bit 15 of the ALU output.

	(2) Q left shift linkage. (3) RAM left shift linkage.
REGISTER	Keyword used in sequence microstatements to identify the sequencer register, as in GO TO REGISTER.
RESETBITMR	Interrupt control microstatement to clear bits in the MR based on information on the data bus.
RESETMEM	Command to clear the memory FF.
RESETMR	Interrupt control microstatement to clear all bits in the MR, which enables all interrupts.
RETURN	Sequence microstatement to return from a subroutine call.
RSBF	Shifted ALU destination, causing the B register to be right shifted by one bit and the result of the ALU operation, after being shifted right by one bit, to be available at the ALU output.
RSQBF	Shifted ALU destination, causing the B register to be right shifted by one bit and the result of the ALU operation, after being shifted right by one, to be stored in the Q register and be available at the ALU output.
SETBITMR	Interrupt control microstatement to set bits in the MR according to information on the data bus.
SETCC	Keyword for a set condition code microstatement to set macro condition codes according to any ccodes that follow this keyword.
SETMCC	Keyword for a set condition code microstatement to set micro condition codes according to any ccodes that follow this keyword.
SETMR	Interrupt control microstatement to set all bits in the MR, inhibiting all interrupts.

SR	Status register in the interrupt control unit, used as: (1) Bus source. (2) Bus destination.
START	Keyword in a sequence micro-statement, identifying the start address set in switches on MIME (e.g., GO TO START).
THEN	Keyword used in a conditional microstatement (e.g., IF CT* THEN LOOP AT FILE).
TO	Keyword used in a sequence micro-statement (e.g., GO TO MAP).
TOGREAD	Command to toggle the read FF.
TRUE	A constant high, 1 or true state, used for: (1) Condition (synonym for 1). (2) Carry in to an add or subtract ALU operation (synonym for 1).
TTYSTATUS	Condition of the status signal of the RS-232 port; high (true) when the port is ready.
UDACA . . UDACF UDACØ . . UDAC9	Auxiliary command used to direct action of the auxiliary circuit board; defined in description of auxiliary board.
UDAFA . . UDAFF UDAFØ . . UDAF9	Auxiliary functions, used as: (1) Bus source. (2) Bus destination. Causes a read or write operation, respectively, to be requested of the auxiliary circuit board. Auxiliary function directs the auxiliary board on what data to read or write; defined in description of auxiliary board.
UDTCB UDTCC UDTCD UDTCE	Conditions available from the auxiliary board; defined in description of auxiliary board.

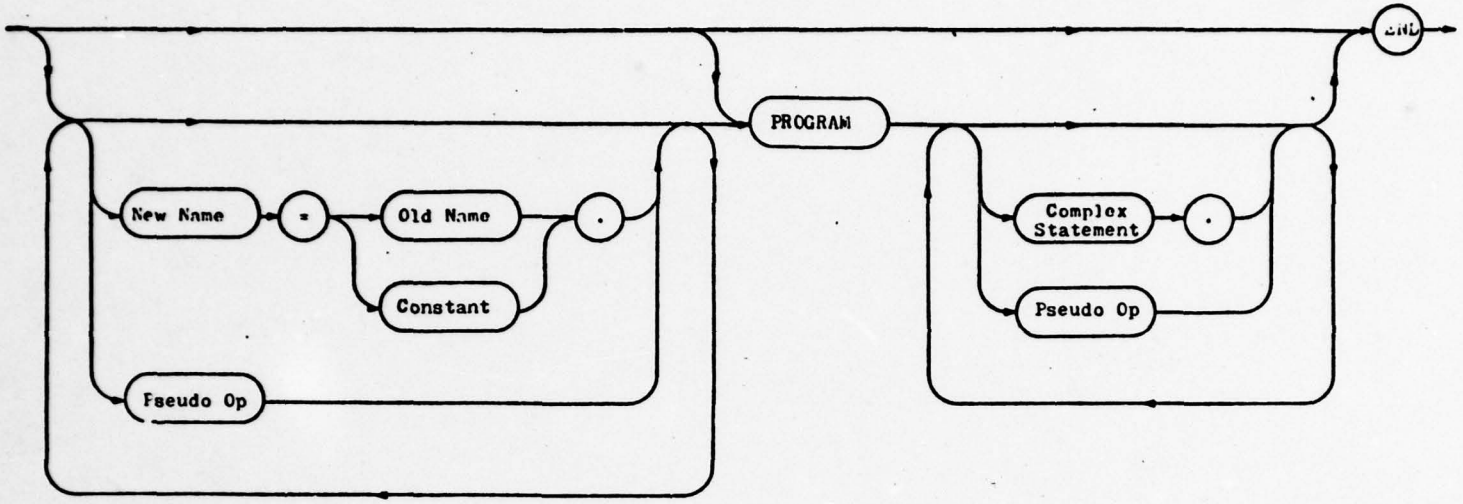
UPCARRY		Ccode for setting the carry condition code from the carry out of the most significant byte of the ALU result.
UPZERO		Ccode for setting the zero condition code based on whether the most significant byte of the ALU result is zero.
V		Branch control for a multi-way branch, based on the status of the macro overflow condition code.
WCR		Word count register, used for: (1) Bus source. (2) Bus destination.
WZERO		Ccode for setting the zero condition code based on whether the full word of the ALU result is zero.
XCHECK	X	Special maintenance programmer pseudo-operation.
XNOR		ALU function.
XOR		ALU function.
Z		Branch control for a multi-way branch, based on the status of the macro zero condition code.
ZERO		(1) Macro zero condition code, used for: (a) Ccode used for setting a condition code based on the status of the macro zero condition code. (b) Condition. (2) Keyword used in a sequence microstatement to signify microaddress zero (e.g., GO TO ZERO).
ZERØ		Ccode to clear a zero condition code.
ZERO1		Ccode to set a zero condition code.
ØF		ALU destination, where result is not stored but is available as

		the ALU output.
ø		A low, ø, or false signal, used as: (1) ALU operand (integer zero). (2) Carry in to an add or subtract ALU function. (3) Shift linkage for Q, RAM, or CENTER with either a left or right shifted ALU destination.
1		A high, 1, or true signal, used as: (1) Condition (synonym for TRUE). (2) Carry in to an add or subtract ALU operation (synonym for TRUE). (3) Q or RAM shift linkage for a shifted ALU destination (both left and right).
2		Label type, signifying that this label will be used for at most a two-way branch.
4		Label type, signifying that this label will be used for at most a four-way branch.
8		Label type, signifying that this label will be used for at most an eight-way branch.
16		Label type, signifying that this label will be used for at most a sixteen-way branch.
+	X	ALU function.
-	X	ALU function.
=	X	Keyword used in an ALU operation (e.g., A OR B = QF).
,	X	Separator used between: (1) Ccodes in a list in a set condition codes micro-statement. (2) Shift linkages with a shifted ALU destination. (3) Branch control elements in a list in a multi-way branch microstatement.
;	X	Delimiter used to terminate each microstatement.

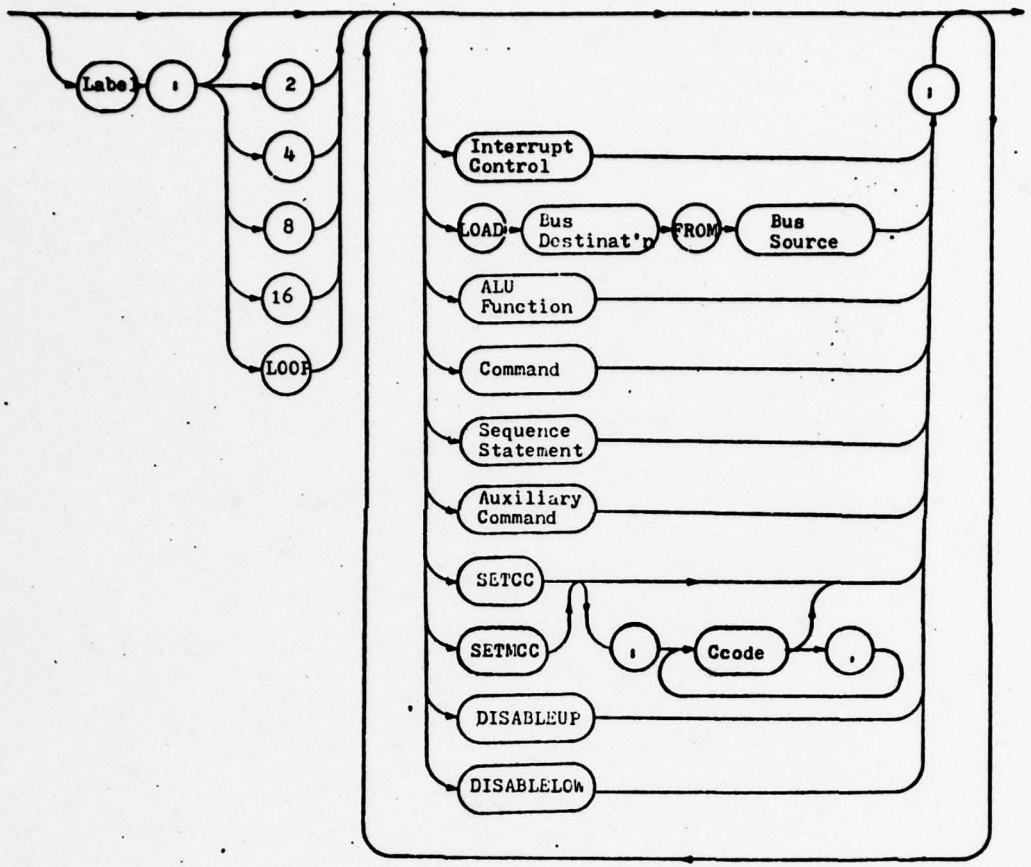
.	X	Delimiter used to terminate a group of microstatements that make up a microinstruction.
:	X	Separator used to: (1) Terminate a label name. (2) Separate ccodes from the set condition codes keyword. (3) Separate shift linkages from shift linkage keyword with a shifted ALU destination.
#	X	Comment delimiter, used to begin and end a comment string.

Attachment C-B

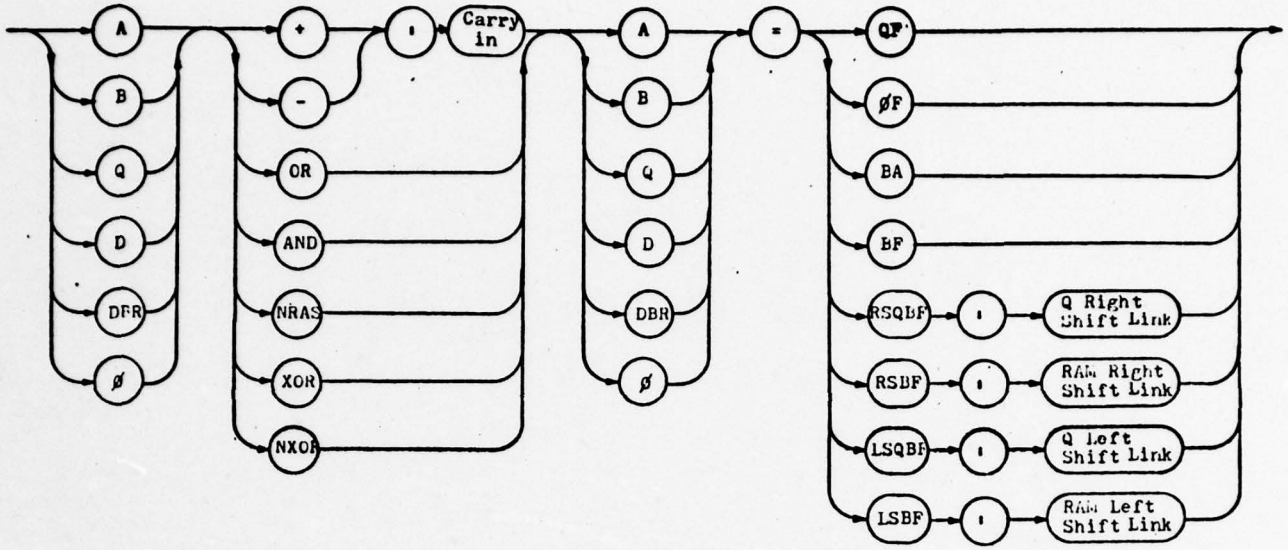
MIME Language Flow Diagrams


PROGRAM
New Name
=
Old Name
Constant
.
Pseudo Op
Complex Statement
.
Pseudo Op

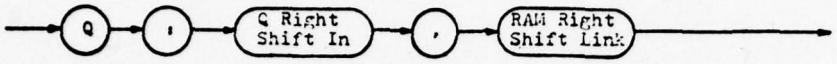
Program



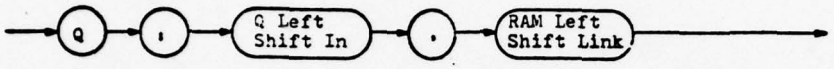
Complex Statement



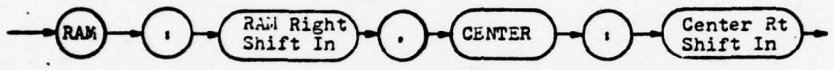
ALU Function



Q Right Shift Link

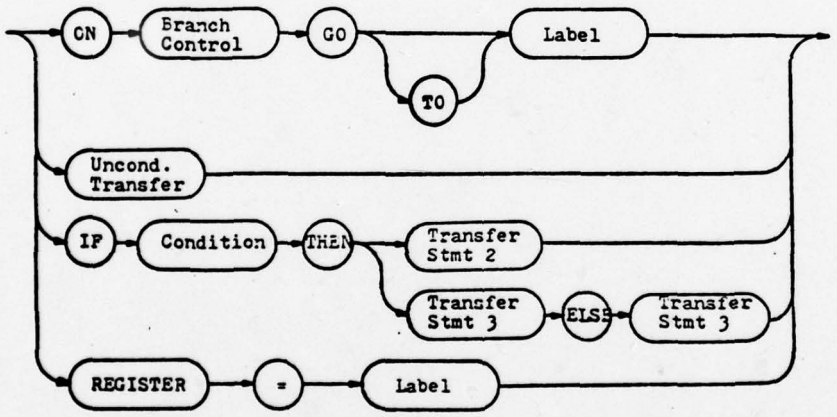


Q Left Shift Link

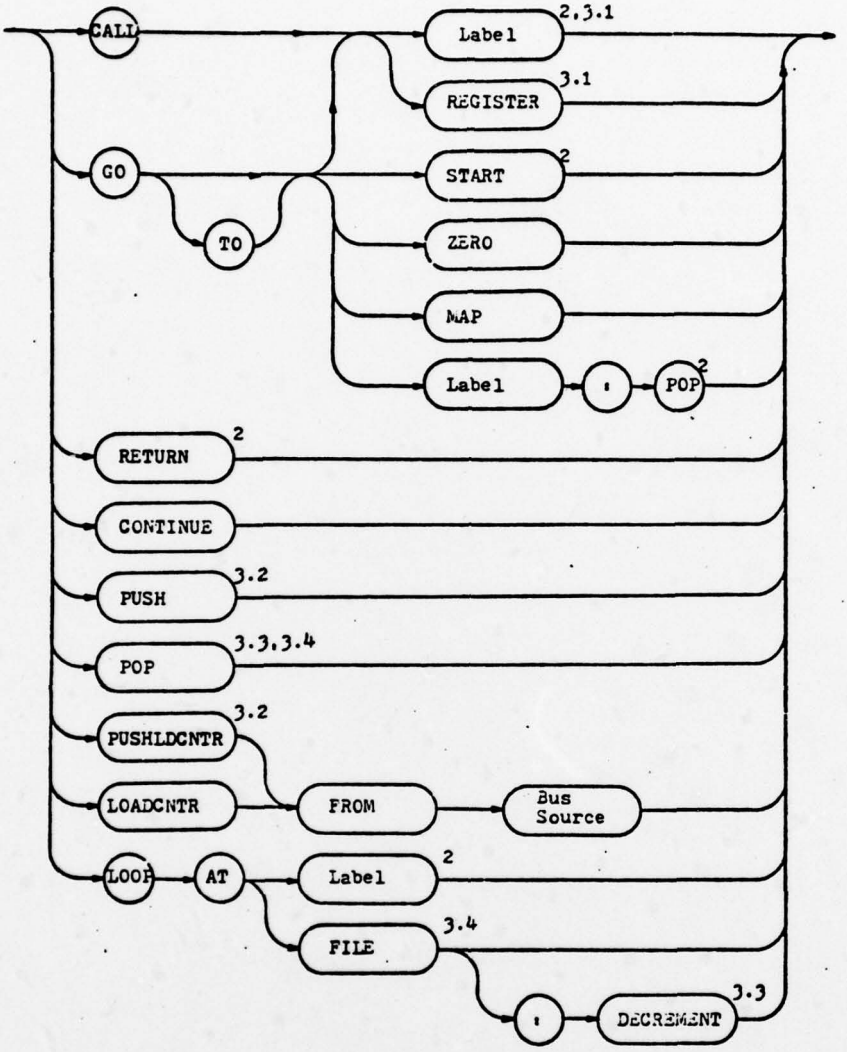


RAM Right Shift Link

RAM Left Shift Link



Sequence Statement



Instructions with superscript 2 qualify as transfer statements 2 for use with the IF-THEN sequence statement.
Instructions with superscript 3 qualify as transfer statements 3 for use with the IF-THEN-ELSE sequence statement. Only similarily numbered instructions may be paired up (e.g., PUSH and PUSHLDCNTR are both numbered 3.2).

Unconditional Transfers

Attachment C-C

Error Messages

This is a list of the error messages that the translator may generate. If the error number is followed by the letter "S", that error will cause the translator to enter a "scan" mode to look for the end of the current microstatement before resuming processing (see section VI of this appendix). Following the text of the error message may be additional information to aid the programmer in determining how to correct the error. Messages of the form "---FIELD ALREADY IN USE" may indicate that two of the same type microstatements are in the same microinstruction, thereby causing the translator to try setting a field in the microword twice. A missing period to terminate a microinstruction is many times the cause of this.

Error Number	Error Message
1S	OLD NAME OF A DEFINITION NOT A RESERVED WORD The word on the right side of the equal sign of a redefinition must be one of the reserved words that can be redefined, as noted in Attachment C-A.
2S	TOKEN FOLLOWING A LABEL NOT A RESERVED WORD The word that starts the first microstatement on a line after a label is not a reserved word.
3S	START OF LINE IN PROGRAM PART NOT A LABEL OR RESERVED WORD First word of a microinstruction was not a label or a valid reserved word that can begin a micro-statement.
4S	INVALID DESTINATION REGISTER IN -LOAD- (See Table CIV)
5S	-FROM- MISSING FROM THE -LOAD- STATEMENT The keyword FROM was not found in a bus transfer microstatement.
6S	INVALID SOURCE REGISTER IN -LOAD- STATEMENT (See Table CIII)
7S	INVALID START OF STATEMENT The first word found when expecting the start of a microstatement was not a valid reserved word to start a microstatement with.
8S	INVALID CONDITION CODES IN A SET CONDITION CODES STATEMENT The ccodes in the microstatement are not valid ccodes (See Table CVII).
9S	INVALID ALU FUNCTION (See Table CX)
10	NO CARRY IN FIELD ON A + OR - ALU FUNCTION All add and subtract ALU operations must have a carry-in field specified (See Table CXI).
11	INVALID CARRY IN NAME (See Table CXI).
12	NRAS ALU FUNCTION HAS INVALID OPERANDS The NRAS ALU function is the only ALU function where the order of the operands is critical (See Figure C1).

13 INVALID ALU OPERAND PAIR
All operand pairs are acceptable except Q-B, DBR-B, and when both operands are the same (See Figure C1).

14S INVALID ALU OPERAND
(See Table CIX)

15S NO EQUAL SIGN IN ALU FUNCTION STATEMENT

16S NO COLON FOLLOWING SHIFTED ALU DESTINATION

17 INVALID SHIFT-IN FIELD
The shift linkage does not have a valid source (See Tables CXIII-CXVIII)

18S NO COLON FOLLOWING -Q-, -RAM-, or -CENTER-
The keyword for a shift linkage must be separated from the shift linkage source by a colon.

19 LABEL ALREADY DEFINED - LATEST DEFINITION WILL BE USED

20 INVALID ALU DESTINATION FIELD
(See Table CXII)

21 CANNOT DO ALU OPERATION AND CONDITIONAL STATEMENT IN SAME MICROINSTRUCTION
Selecting the conditions on a conditional statement and setting the condition codes on an ALU operation both use some of the same fields in the microword. Note that some of the unconditional transfer microstatements are actually conditional microstatements with a constant true or false condition (See Section IV).

22S COMMENT NOT ALLOWED HERE
Comments are only acceptable after the period that terminates a redefinition or microinstruction, or starting on a new line.

23 INVALID BRANCH CONTROL FIELD ON A MULTI-WAY BRANCH STATEMENT
The branch control field may only be selected from those bits specified in section IV, and can only be grouped as shown in that section.

24S NO -GO TO- STATEMENT PART OF MULTI-WAY BRANCH
A sequence microstatement that began with the keyword ON did not have a GO TO <label> as part of the microstatement.

25S UNDEFINED LABEL IN MULTI-WAY BRANCH

26S INVALID START OF EXPECTED TRANSFER STATEMENT
The translator did not find a transfer microstatement when the microprogram structure indicated one should be present.

27S ILLEGAL TO REDEFINE -PROGRAM-, -END-, OR PSUEDO OPS
These keywords cannot be redefined (See Attachment C-A).

28S NO -THEN- IN AN -IF- STATEMENT

29 INVALID TRANSFER STATEMENT PAIR IN AN IF-THEN-ELSE CONSTRUCT
Section IV defines which unconditional microstatements may be paired together in the IF-THEN-ELSE construct.

30 INVALID TRANSFER STATEMENT IN IF-THEN CONSTRUCT
Section IV defines which unconditional microstatements may be used in the IF-THEN construct.

31 INVALID CCODE
(See Table CVII)

32 EOF ENCOUNTERED WITH NO -END- CARD FOUND
The end of file was encountered unexpectedly.

33 SEQUENCER INSTRUCTION FIELD OR CONDITION CODE FIELD ALREADY IN USE
In trying to set up a sequence microstatement, the fields for the sequencer instruction or the field for selecting a condition were already in use. Condition code fields are also used for a set condition code microstatement, multi-way branch microstatements, and constants.

34S INVALID START OF STATEMENT
See error 7.

35 INTERRUPT CONTROL FIELD ALREADY IN USE
Auxiliary commands also use this field, as do multi-way branches using the interrupt vector and bus transfers using the SR or MR as a source or destination.

36 MULTIPLE USE OF DATA BUS
Only one source can be gated onto the data bus at any one time. CLEARMR interrupt control microstatement also uses the data bus.

37 COMMAND FIELD ALREADY IN USE
Check for another command microstatement in the

same microinstruction.

38 BUS DESTINATION FIELD ALREADY IN USE
Multiple bus transfer microstatements can be specified in a microinstruction as long as the different destinations are set using different fields in the microword and there is only one bus source specified.

39 AUX FUNCTION FIELD ALREADY IN USE
Check for other microstatements in this micro-instruction that reference an auxiliary function.

40 BUS SOURCE FIELD ALREADY IN USE

41 INVALID HEX CONSTANT
Constants must consist of one to four hexadecimal characters (Ø through 9 and A through F) enclosed in apostrophes.

42 CONSTANT FIELD(S) ALREADY IN USE
The fields which make up the constant are also used for the branch control of the multi-way branch, condition selection for conditional and uncon-ditional sequence microstatements, and selecting the source to set condition codes by.

43 SETCC OR SETMCC ALREADY IN USE
Cannot set both macro and micro condition codes in the same microinstruction.

44 NEG MUX FIELD ALREADY USED
The source for setting the negative condition code has already been specified.

45 OVR MUX FIELD ALREADY USED
The source for setting the overflow condition code has already been specified.

46 ZERO MUX FIELD ALREADY USED
The source for setting the zero condition code has already been specified.

47 CARRY MUX FIELD ALREADY USED
The source for setting the carry condition code has already been specified.

48 UPPER-BYTE-DISABLE FIELD ALREADY USED
DISABLEUP has already been specified in this micro-instruction.

49 LOWER-BYTE-DISABLE FIELD ALREADY USED
DISABLELOW has already been specified in this

microinstruction.

50 ALU FUNCTION FIELD ALREADY IN USE

51 CARRY-IN FIELD ALREADY IN USE

52 ALU DESTINATION FIELD ALREADY IN USE

53 CENTER MUX FIELD ALREADY IN USE
This field is used to set the value specified by the CENTER shift linkage on a shifted ALU destination. A branch address in the microinstruction also uses this field.

54 RMUX FIELD ALREADY IN USE
This field is used to set the value specified by the RAM: shift linkage on a shifted ALU destination. A branch address in the microinstruction also uses this field.

55 QMUX FIELD ALREADY IN USE
This field is used to set the value specified by the Q: shift linkage on a shifted ALU destination. A branch address in the microinstruction also uses this field.

56S INVALID ALU SHIFTED DESTINATION
(See Table CXII)

57 CAN ONLY -CALL- A LABEL OR -REGISTER-
In a subroutine call, only a label or the keyword REGISTER can be used as the subroutine identifier (See Section IV). This error may also generate error 58.

58 UNDEFINED LABEL

59 -GO TO <LABEL> :- NOT FOLLOWED BY -POP-
The microstatement GO TO <label> followed by a colon can only be followed by the keyword POP (See Section IV).

60 ILLEGAL -GO TO- DESTINATION
Only valid destinations on a GO TO microstatement are a <label> , REGISTER, MAP, START, or ZERO (See Section IV).

61 -LOOP AT FILE:- NOT FOLLOWED BY -DECREMENT-
The conditional microstatement LOOP AT FILE followed by a colon can only be followed by the keyword DECREMENT (See Section IV).

62 -LOOP AT- NOT FOLLOWED BY <LABEL> OR -FILE-
A LOOP AT conditional microstatement can only have a <label> or FILE as its destination (See Section IV).

63S MORE THAN 4 BRANCH CONTROLS ON A MULTI-WAY BRANCH

64S INVALID GROUPING OF BRANCH CONTROLS
The branch control field must follow the restrictions outlined in Section IV).

65 BRANCH CONTROL TYPE AND LABEL TYPE NOT COMPATIBLE ON MULTI-WAY BRANCH
The number of branch control elements in the branch control field dictates the minimum label type required. The label that is the destination of the multi-way branch is not declared at or above this minimum label type (See Section IV).

66 POLARITY FIELD ALREADY IN USE
This field is used for the keyword NOT in a condition, many conditional sequence microstatements (depending on the order of the unconditional microstatements in the conditional one), some unconditional sequence microstatements (CALL REGISTER, GO TO REGISTER, PUSH and all LOOP AT microstatements), for setting the overflow flag on a set condition codes microstatement, multi-way branches, and constants.

67 ALU CONDITION FIELD ALREADY IN USE
The ALU condition selector field (See Table CXX) is also used for setting the carry flag on a set condition codes microstatement, for selecting I/O conditions (See Table CXIX) and for constants.

68 IO CONDITION FIELD ALREADY IN USE
See error 67

69 CCU CONDITION FIELD ALREADY IN USE
The CCU condition selector field (also referred to as miscellaneous conditions, (See Table CXXI) is also used for setting the zero flag on a set condition codes microstatement, for multi-way branches, and for constants.

70S INVALID CONDITIONS
(See Tables CXIX, CXX, and CXXI)

71S INVALID START OF -IF-, -ON-, OR UNCONDITIONAL TRANSFER STATEMENT
Refer to Section IV for permissible starting keywords for sequence microstatements.

72S END OF STATEMENT FOUND WITH NO SEMICOLON
A semicolon must terminate each microstatement, even if it is the last microstatement before a period that terminates a microinstruction.

73 FIELD(S) FOR MICRO-BRANCH-ADDRESS ALREADY IN USE
The micro branch address fields are also used for selecting the shift linkages in a shifted ALU destination.

74 MEMORY TRANSFER ONLY VALID TO OR FROM MBR
If MEMORY is one of the operands of a bus transfer microstatement, the other operand must be either MBR or MBR/DMA (See Section IV).

75 ALU OPERAND FIELD ALREADY IN USE

76 AUX COMMAND FIELD ALREADY IN USE
See error 35.

77 ILLEGAL TO ATTEMPT TO REDEFINE A DELIMITER
Certain characters (comma, period, colon, equal sign, minus sign, plus sign, semicolon, and the comment character, #) cannot be redefined (See Attachment C-A).

78S INVALID RESERVED WORD IN REDEFINITION SECTION
When looking at the start of a redefinition, the only reserved words acceptable are END, PROGRAM, pseudo-operations or the start of a comment.

79S REDEFINITION NOT TERMINATED BY A PERIOD

80S NON RESERVED WORD NOT FOLLOWED BY AN EQUAL SIGN IN REDEFINITION SECTION

81 -ORG- NOT FOLLOWED BY A CONSTANT

82S NO -FROM- FOLLOWING THE LOAD COUNTER INSTRUCTION
Both LOADCNTR and PUSHLDCNTR must be followed by a FROM <bus source> phrase.

83S INVALID SOURCE REGISTER FOR LOADING COUNTER
The bus source register must be one of those named in Table CIII.

84 FIELD(S) FOR MICRO LOOP COUNTER ALREADY IN USE
This error message is not used currently.

85 -REGISTER- NOT FOLLOWED BY AN EQUAL SIGN
A microstatement starting with the keyword REGISTER must be followed by an equal sign and a label name.

86 NO CENTER SHIFT IS SPECIFIED IN SHIFTED ALU DESTINATION

87 NO Q SHIFT IS SPECIFIED IN SHIFTED ALU DESTINATION

88 NO RAM SHIFT IS SPECIFIED IN SHIFTED ALU DESTINATION

89 WARNING - REGISTER NOT EXPLICITLY FILLED IN PREVIOUS MICROINSTRUCTION
When a microstatement uses the keyword REGISTER as a destination (as in GO TO REGISTER), the value in the register is taken from the previous microinstruction's microbranch address field. If the previous microinstruction did not contain a " REGISTER = ----" microstatement, this warning message is printed. It is possible, even though there was no "REGISTER = ----" microstatement in the previous microinstruction, that the microbranch address field in the previous microinstruction was filled by some other microstatement (such as a GO TO <label> or CALL <label>). The programmer should be certain that this is the desired result.

Attachment C-D

MIME Language Definition

MIME LANGUAGE DEFINITION
---- -------- ----------

NOTATION:

```
:=      ='IS DEFINED AS'
<     > = NON-TERMINAL SYMBOL
[     ] = OPTIONAL
(     ) = SELECT ONE
(    )+ = SELECT ONE OR MORE
!       = OR
```


MIME LANGUAGE DEFINITION
---- -------- ----------

<CHARACTER> := ANY ALPHABETIC CHARACTER A..Z ! ANY NUMERIC CHARACTER
0..9 ! ANY SPECIAL CHARACTER EXCEPT # SPACE . , ; : + - = '

<HEX CHARACTER> := CHARACTERS 0..9 ! CHARACTERS A..F

<PROGRAM> := <REDEFINITIONS> PROGRAM <COMPLEX STATEMENT> END

<REDEFINITIONS> := <EMPTY> ! <COMMENT> ! <REDEFINITIONS> <PSEUDO
OP> ! <REDEFINITIONS> <NEW NAME> = <OLD NAME> . [<COMMENT>]
! <REDEFINITIONS> <NEW NAME> = <CONSTANT> . [<COMMENT>]

<COMPLEX STATEMENT> := <EMPTY> ! <COMMENT> ! [<LABEL>:[<LABEL TYPE>]]
<MICROINSTRUCTION>.<COMMENT><COMPLEX STATEMENT> ! <PSEUDO OP>
<COMPLEX STATEMENT>

```
<MICROINSTRUCTION> := <MICROSTATEMENT>; !
    <MICROSTATEMENT>;<MICROINSTRUCTION> !
    <EMPTY>

<MICROSTATEMENT> := <EMPTY> ! <INTERRUPT CONTROL> ! <BUS TRANSFER> !
    <ALU FUNCTION> ! <COMMAND> ! <SEQUENCE STATEMENT> !
    <SET CONDITION CODES> ! <DISABLE BYTE> ! <AUXILIARY FUNCTION>

<LABEL> := <1 TO 79 CHARACTERS>

<LABEL TYPE> := 2 ! 4 ! 8 ! 16 ! LOOP

<ALU FUNCTION> := <ALU SRC><ALU FN>[:<CARRY IN>]<ALU SRC>=(<ALU DEST> !
    <ALU SHIFT DEST>)

<ALU SRC> := A ! B ! Q ! D ! DBR ! 0

<ALU FN> := + ! - ! OR ! AND ! NRAS ! XOR ! XNOR

<ALU DEST> := QF ! OF ! BA ! BF

<COMMAND> := INCPC ! INCMAR ! INCDMA ! TOGREAD ! RESETMEM ! DCDLOWIR !
    IOWRITE ! IOREAD ! HALT

<BUS TRANSFER> := LOAD <DESTINATION> FROM <SOURCE>

<DESTINATION> := PC ! MAR ! MBR ! IR ! MEMORY ! DBR ! D ! CCR ! BAR !
    WCR ! FP ! IOBR ! DB ! MR ! SR ! ALATCH ! BLATCH ! ABLATCH !
    <AUX FUNCTION> ! MBR/DMA)

<SOURCE> := PC ! MBR ! MBR/DMA ! IR ! MEMORY ! ALU ! CCR ! WCR ! FP !
    IOBR ! MR ! SR ! <CONSTANT> ! <AUX FUNCTION>
```

```
<CONSTANT> := '<UP TO 4 HEX CHARACTERS>'

<SEQUENCER STATEMENT> := <MULTIWAY BRANCH> ! <CONDITIONAL STATEMENT> !
    <UNCONDITIONAL STATEMENT> ! REGISTER = <LABEL>

<MULTIWAY BRANCH> := ON <BRANCH CONTROL> GO TO <LABEL>

<BRANCH CONTROL> := (Z ! N ! C ! V)+ !
    (IV3 ! IV2 ! IV1 ! IV0)+ !
    (IR15 ! IR14 ! IR13 ! IR12)+ !
    (IR12 ! IR11 ! IR10 ! IR9)+ !
    (IR9 ! IR8 ! IR7 ! IR6)+ !
    (IR6 ! IR5 ! IR4 ! IR3)+ !
    (IR3 ! IR2 ! IR1 ! IR0)+

<SET CONDITION CODES> := (SETCC ! SETMCC)[:<CONDITION CODE LIST>]

<CONDITION CODE LIST> := <EMPTY> ! <CONDITION CODE LIST>,<CCODES>

<DISABLE BYTE> := DISABLELOW ! DISABLEUP

<AUX FUNCTION> := UDAF0 ! UDAF1 ! UDAF2 ! UDAF3 ! UDAF4 ! UDAF5 !
    UDAF6 ! UDAF7 ! UDAF8 ! UDAF9 ! UDAFA ! UDAFB ! UDAFC !
    UDAFD ! UDAFE ! UDAFF

<NEW NAME> := <LABEL>

<OLD NAME> := <ANY RESERVED WORD>
    NOTE:  SEE LIST OF RESERVED WORDS- ATTACHMENT C-A

<UNCONDITIONAL STATEMENT> := <TRANSFER STATEMENT>
```

```
<TRANSFER STATEMENT> := PL TRANSFER STATEMENT> ! <R TRANSFER STATEMENT>
    ! CONTINUE ! LOADCNTR FROM <SOURCE> ! GO TO MAP ! GO TO ZERO

<PL TRANSFER STATEMENT> := CALL <LABEL> ! GO TO <LABEL> ! GO TO START !
    LOOP AT <LABEL> ! RETURN ! GO TO <LABEL>:POP

<R TRANSFER STATEMENT> := PUSH ! CALL REGISTER !
    PUSHLDCNTR FROM <SOURCE> ! CALL <LABEL> ! GO TO REGISTER !
    GO TO <LABEL> ! POP ! LOOP AT FILE ! LOOP AT FILE:DECREMENT

<ALU SHIFT DEST> := RSQBF:Q:<Q RIGHT SHIFT IN>,RAM:<RAM RIGHT SHIFT IN>
    ,CENTER:<CENTER RIGHT SHIFT IN> !
    RSBF:RAM:<RAM RIGHT SHIFT IN>,CENTER:<CENTER RIGHT SHIFT IN> !
    LSQBF:Q:<Q LEFT SHIFT IN>,RAM:<RAM LEFT SHIFT IN>,CENTER:<CENTER
    LEFT SHIFT IN> !
    LSBF:RAM:<RAM LEFT SHIFT IN>,CENTER:<CENTER LEFT SHIFT IN>

<Q RIGHT SHIFT IN> := CARRY ! DBR15 ! Q0 ! RAM0 ! RAM8 ! 0 ! 1

<RAM RIGHT SHIFT IN> := CARRY ! DBR15 ! Q0 ! RAM0 ! RAM8 ! 0 ! 1

<CENTER RIGHT SHIFT IN> := CARRY ! DBR7 ! RAM8 ! 0

<Q LEFT SHIFT IN> := CARRY ! Q15 ! RAM15 ! 0 ! 1

<RAM LEFT SHIFT IN> := CARRY ! Q15 ! Q7 ! RAM15 ! RAM7 ! 0 ! 1

<CENTER LEFT SHIFT IN> := CARRY ! RAM7 ! 0
```

```
<CONDITIONAL TRANSFER STATEMENT> := IF <CONDITION> THEN <PL TRANSFER
    STATEMENT> ! IF <CONDITION> THEN <R TRANSFER STATEMENT> ELSE
    <R TRANSFER STATEMENT>

<CONDITION> := [NOT] (BP1 ! BP2 ! MAR0 ! MEMORY ! 1 ! CT* ! IR4 ! IR5 !
    IR7 ! IR15 ! DMAOVER ! DMATERM ! DMAREAD ! TTYSTATUS ! IRQ* ! TRUE
    MF0 ! MALTB ! MAGTB ! MCARRY ! MOVER ! MNEG ! MZERO ! F0 !
    ALTB ! AGTB ! CARRY ! OVER ! NEG ! ZERO)

<CCODES> := BNEG ! BOVER ! LOWZERO ! UPZERO ! WZERO ! NEG ! OVER !
    ZERO ! CARRY ! CARRY* ! AC1 ! LOWCARRY ! UPCARRY ! RAM0 !
    RAM15 ! Q15 ! RAM7 ! RAM8 ! NEG0 ! NEG1 ! OVER0 ! OVER1 !
    CARRY0 ! CARRY1 ! ZERO0 ! ZERO1

<CARRY IN> := RAM7 ! MCARRY ! MCARRY* ! CARRY ! CARRY* ! 0 ! 1 !
    TRUE ! FALSE

<INTERRUPT CONTROL> := CLEARMASTR ! CLEARALL ! CLEARDB ! CLEARMR !
    CLEARVEC !  SETMR ! RESETMR ! RESETBITMR ! SETBITMR !
    DISABLEINT ! ENABLEINT

<PSEUDO OPERATION> := ORG <CONSTANT> ! PAGE <TITLE> ! PASS3 ! DEBUG
    ! XCHECK

<TITLE> := <UP TO 50 CHARACTERS>

<COMMENT> := #<ANY NUMBER OF CHARACTERS>#
```

Appendix D

MIME Modest Monitor (MIME/MM) User Manual

Preface

As the MIME project progressed into the emulation of a full minicomputer instruction set, it became apparent that a way was needed to load and dump registers and memory and to monitor program execution. This could be done through the MIME front panel for most registers and memory, but when auxiliary registers were added there was no way to directly load or read them. Furthermore, front panel entry and dump of data was slow and error prone.

To fill this need the MIME Modest Monitor was written. Although modest in its capabilities, this program can greatly speed the debugging of both macro and microprograms. It was designed to embody only the most vital features, and to be easily interfaced either to a microcomputer acting as a controller or to a terminal for direct use. In the former case, the external computer could be programmed to interact with the MIME Modest Monitor and to provide more complex features. In the latter case, the operator must bear with the limitations of the program, but it is hoped that it will be as useful a tool to others as it has been to the authors.

N.B. It is assumed in this manual that the user is familiar with the structure of MIME, including the names of the

registers and basic MIME operation. This information is available in Appendix A, MIME User's Manual.

Contents

List of Figures

List of Tables

MIME Modest Monitor User Manual

This manual describes the MIME Modest Monitor (MIME/MM) and gives instructions for its operation. The MIME/MM is a minimum-capability interactive monitor for use of the Microprogrammable Minicomputer Emulator (MIME). The monitor program is microcoded and resides in MIME control store from address 000_{16} to $03F_{16}$. It is provided on a set of eight 2708 EPROMs on the CS1 board. MIME/MM also uses main memory from address $03D0_{16}$ to $03F8_{16}$ as working space.

Features

Using the MIME/MM, a user may dump and load MIME registers, dump and load memory locations, dump and load paper or cassette tapes, and begin execution of a user program at any location in memory.

MIME/MM is designed to run on any type of terminal connected to the MIME RS-232 serial input/output port, using baud rates from 110 to 19,200 bits per second. The program operates in the full duplex mode and echoes all input characters.

The source code (in MIME translator language) for all routines is available in Volume III.

Structure of the Program

An overall flow chart of the MIME/MM appears in Figure D1. When the monitor is invoked, the program prints "MIME/MM" on the terminal and stacks the MIME registers in memory at the locations listed in Table DI. The program then proceeds to the Command routine which prints a prompt on the terminal, accepts a command,

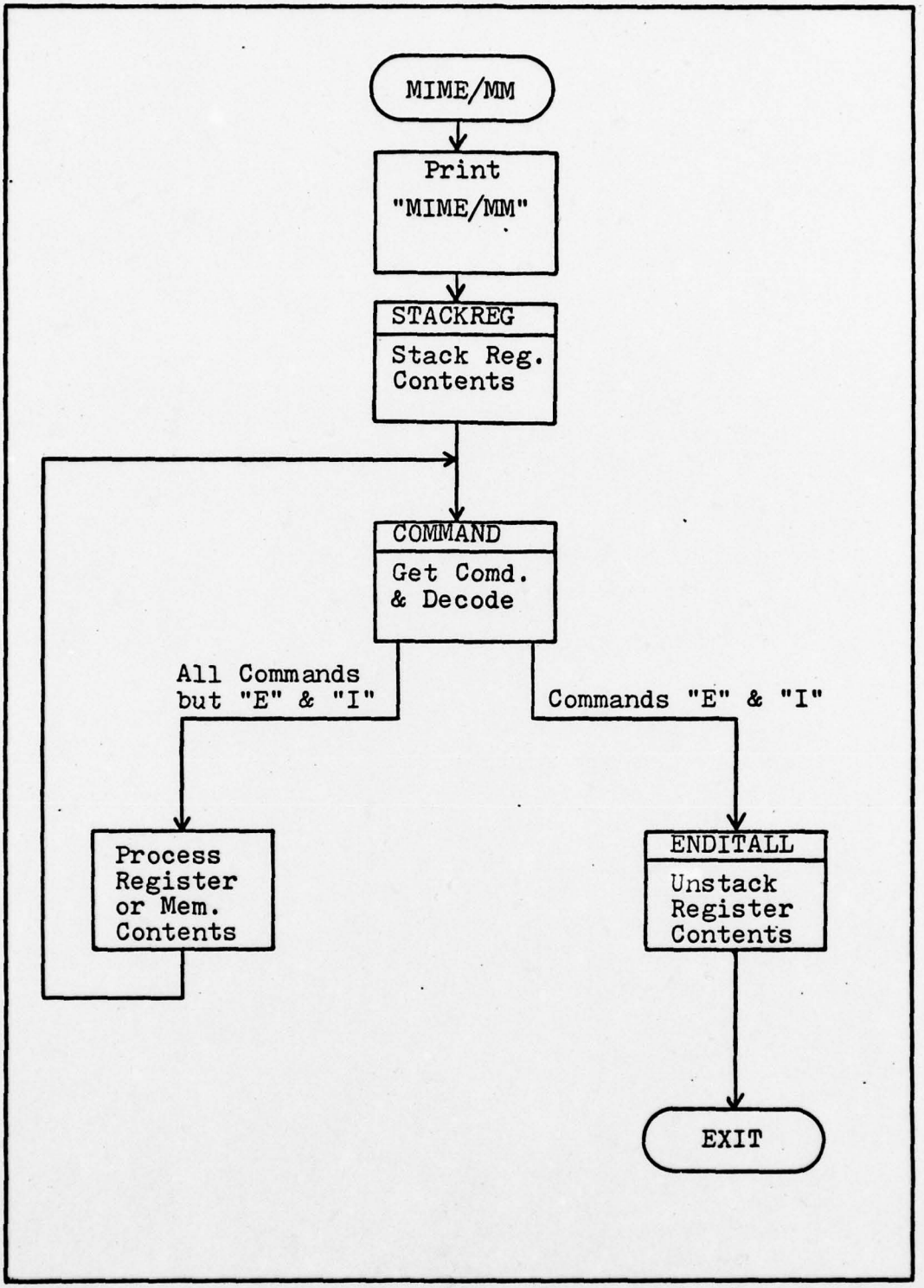


Figure D1. MIME/MM Flow Chart

Table DI

Memory Stack Addresses

Register Name	Stack Addr.
Memory Buffer Register (MBR)	03D0
Data Buffer Register (DBR)	03D1
Q Register	03D2
A Register	03D3
B Register	03D4
General Purpose Register 0	03D5
:	:
F	03E4
Program Counter (PC)	03E5
Instruction Register (IR)	03E6
Condition Code Register (CCR)	03E7
Mask Register (MR)	03E8
User Definable Function 0	03E9
:	:
F	03F8

decodes the command, and branches to the appropriate service routine. Upon completion of the service routine, the program returns to issue a new command prompt. If an error is encountered in any routine, an error message is printed and the program returns to issue a new command prompt. The Exit command unstacks the MIME register values from memory, inserts the values into the MIME registers and branches to the macro instruction fetch routine to begin execution of the macro level program.

Invoking the MIME/MM

The MIME/MM program begins at control store (microaddress location) 000_{16} hence any method of beginning microprogram execution at address 000 may be used. Normally, when MIME is reset, the microaddress is set to 000 so that one need only depress the RUN switch to begin execution of the monitor. MIME/MM will issue the message

MIME/MM

COM?

The first line indicates that the program is at the entry point and the "COM?" prompt notifies the user that MIME/MM is awaiting an input command.

Another method of entering MIME/MM programs is to assign a spare machine operation code to cause a branch to microaddress 000. This is easily accomplished by properly programming the MIME mapping PROMS. This allows the user to execute machine level programs up to the point where the register and/or memory contents need to be observed, branch the MIME/MM, examine/change the information and branch back to the program to continue

execution.

Commands

The following pages describe the commands recognized by the MIME/MM and give examples of their use. In all examples, the operator input is underlined while the computer output is not. Also, all operator input must be followed by a carriage return, unless noted.

FUNCTION - Dump Register Contents

COMMAND - DR

ACTION - The values of MIME's registers at time of MM invocation are read from the memory stack and displayed on the terminal.

NOTES - 1. Labels for the registers are as follows:

D	Data Buffer Register
Q	ALU Q Register
A	ALU Register currently addressed by A Latch
B	ALU Register currently addressed by B Latch
GEN PURP	ALU General Purpose Register (R0-RF)
MB	Memory Buffer Register
P	Program Counter
I	Instruction Register
C	Condition Code Register
MR	Mask Register on Am2914 Interrupt Control Unit
UDAF	User Definable Functions - if not defined as registers, contents will be "FFFF"

2. The contents of the MAR, BAR, and A and B latches are not available to the DB and are not displayed by MIME/MM.

3. The contents of the IOBR and WCR are not displayed by MIME/MM.

EXAMPLE OF USE -

```
MIME/MM
COM? DR (cr)
REGISTERS
D=0000 Q=00F8 A=C0F8 B=0000
GEN PURP
0  0000 C8FC C83C C8FC C03C C8FC C03C 00F0
8  C83C C8FC C83C C8F8 C038 C8FC 0000 C0F8

MB=0000 P=0000 I=FFFF C=FF10 MR=FF3F
UDAF
0  FFFF 0000 0000 0000 0000 0000 0000 0000
8  0001 4000 0000 FFFF FFFF FFFF FFFF FFFF

COM?
```

FUNCTION - Load Register with New Value

Command - LR

ACTION - The program responds to the command with "REG?" as a prompt for a register name. The operator must type the appropriate abbreviated register name on the terminal. Allowed names are:

D	Data Buffer Register
Q	Q register in ALU
RO...RF	General Purpose Registers in ALU
MB	Memory Buffer Register
P	Program Counter
I	Instruction Register
C	Macro Condition Code Register
MR	Mask Register in Am 2914 Interrupt Control Unit
UO...UF	User Definable Functions if defined as registers

An ESC entered from the terminal will return the program to the "COM?" prompt. Any other input from the terminal will cause the program to simply repeat the "REG?" prompt.

Upon receipt of a valid register name, the monitor responds with"VAL?" requesting the value to go in the register. Four valid hexadecimal characters may be entered. Any invalid character will generate an error and return to the "COM?" prompt. After receiving four valid hexadecimal characters, the

program loads the value into the register stack in memory and returns to the "REG?" prompt to accept further register changes. An ESC will terminate the LR command and return to the "COM?" prompt for a new command.

NOTES - 1. This command only changes values in the register stack in memory. Actual loading of the registers does not take place until an exit command is executed.

2. If an error is encountered, any valid changes made prior to that error are retained. Only the register change in progress at the time of the error is lost.

3. Upon receipt of an error, LR may be reentered at any time to complete any further desired register changes.

4. The MAR, BAR, WCR, and IOBR cannot be loaded through the MIME/MM.

EXAMPLE OF USE -

```
COM? LR (cr)
REG? P (cr)
VAL? 1234 (cr)
REG? R1 (cr)
VAL? 111H
***ERROR
COM? LR (cr)
REG? R1 (cr)
VAL? 111F (cr)
REG? (esc)
COM?
```

FUNCTION - Load Memory With Successive Desired Values

COMMAND - LM

ACTION - In response to the command, the MIME/MM issues an "ADD?" prompt. The user should enter the four digit hexadecimal address where memory loading is to start. The program responds with a space and waits for the desired data for the location to be entered, again as a four digit hexadecimal number. No carriage return is required or allowed at this point. The program repeats this space-wait cycle for sequential memory locations until either an ESC or an invalid hex character is entered. The former returns the program to the "COM?" prompt while the latter causes an error followed by the "COM?" prompt.

NOTES - 1. Changes are made in memory as the data is entered.

2. Any data entered prior to an error will be retained in memory.

3. LM may be reentered at any time prior to the E command.

4. Care should be taken not to overwrite the register stack located at $03D0_{16}$ - $03F8_{16}$. No error is generated when this occurs.

5. No error is generated if an attempt is made to load memory not physically present -- the data is lost.

EXAMPLE OF USE - See example for Dump Memory.

FUNCTION - Dump Desired Memory Locations

COMMAND - DM

ACTION - In response to this command, MIME/MM issues a carriage return and line feed followed by an "ADD?" prompt. The user then enters the four digit hexadecimal address at which the dump is to begin. The program prints the contents of that location on a new line and waits for user input. At this point an ESC will terminate the command and return to the "COM?" prompt for a new command. A space will cause the contents of the next consecutive memory address to be printed. Any other character will be ignored. Memory location contents are printed 16 words to a line.

NOTES - 1. Any invalid character in the address response will cause an error and return to the "COM?" prompt.

EXAMPLE OF USE -

COM? LM (cr)
ADD? 0000 (cr)
0000 1111 2222 3333 4444 5555 6666 7777 (esc)
COM? DM (cr)
ADD? 0000 (cr)
0000_1111_2222_3333_4444_5555_6666_7777_F8C3 (esc)
COM?

FUNCTION - Dump to Tape

COMMAND - DT

ACTION - In response to this command, MIME/MM prompts the user with "ADD?", a request for the address in memory at which the data dump will begin. The operator should respond with the four-digit hexadecimal address. The monitor then requests the number of words to be dumped. The user response is a four-digit hexadecimal number, only the last two (least significant) digits of which are used. Hence, one dump may consist of 0_{10} ($XX00_{16}$) to 255_{10} ($XXFF_{16}$) words. Following the carriage return MIME/MM will dump the contents of the designated number of consecutive memory words to the output port, with a carriage return and line feed separating the data words.

NOTES - 1. The dump routine has been used with 110 Baud teletype (paper tape) and a 300 Baud Silent 700 (TM) terminal with magnetic tape drives.

FUNCTION - Load From Tape

COMMAND - LT

ACTION - The response to this command is the same as for DT. After the number of words is entered, the program waits for input. The tape should be started and data will be loaded into memory and printed on the terminal.

NOTES - 1. This routine ignores all control characters, hence tapes produced on a machine that inserts control characters (e.g. line feed, carriage return, null) between data wards are readable.

2. The loading process must be monitored. After loading the designated number of words, MIME/MM returns to the "COM?" prompt. Anything read from tape except valid commands will produce a stream of errors. The tape should be stopped at that point.

FUNCTION - Exit Monitor

COMMAND - E

ACTION - The register stack is read from memory and the data is loaded into the appropriate MIME registers. Then a branch is made to the machine level instruction fetch routine. This begins execution of the macro program at the address in the Program Counter after the E command.

NOTES - 1. The instruction fetch routine is user supplied with each emulation. Hence, the branch address in the E command must be changed to agree with the emulation currently being performed. This is done by changing the IFTCH address in the the redefinition section of the MIME/MM source program (see Volume III) and retranslating the monitor program.

FUNCTION - Branch to Initialization Routine

COMMAND - I

ACTION - Same as for E except that control passes to the address provided as INITIALADD in the MIME/MM source program. This address may be used as the entry point for an initialization routine for MIME.

Additional Routines

There are two additional routines included in MIME/MM that do not actually provide any monitor function but which have been used as demonstration and debugging aids. They are the Banner and the Memory Diagnostic.

Banner. Typing a B while in the MIME/MM command mode will cause "MIME" to be printed in block letters on the terminal. This routine was used as a demonstration device and as a vehicle for debugging various portions of the MIME hardware. Because of the use of special cursor control characters, this routine will only operate correctly with the Lear-Siegler ADM3 video terminal.

Memory Diagnostic. Another debugging aid included in the MIME/MM PROM is a micro coded memory diagnostic. This routine tests macro memory with various patterns and upon completion displays the highest valid memory address. To run the diagnostic the following steps are necessary:

1. Press RESET.
2. Select ROM position of ROM/RAM switch on CS1.
3. Select MAR as Macro Display.
4. Select A as Micro Display.
5. Select MICRO on MACRO/MICRO selector.
6. Enter the diagnostic's start address.
7. Press RUN.
8. The HALT light should extinguish momentarily and the RUN light should illuminate momentarily.
9. At program completion, Macro Display shows the

highest memory address which is operating correctly. Note that this routine effectively tests parts of the MEM, ALU, CCU1, CCU2, and CS1 modules as well as some features of the Front Panel.